VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Přehled a implementace algoritmů pro detekci komunit v komplexních sítích.

Overview and Implementation of Algorithms for Communities Detection in Complex Networks.

2014 Bc. Lukáš Režnar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Režnar**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Přehled a implementace algoritmů pro detekci komunit v komplexních sítích.
Overview and Implementation of Algorithms for Communities Detection in Complex Networks.

Zásady pro vypracování:

Tato práce se zaměřuje na metody detekce komunit v komplexních sítích, které jsou založeny na metodě random walk (RW). Cílem práce je vytvořit přehled různých možností využití metody RW a implementovat vybrané algoritmy nad různě rozsáhlými datovými kolekcemi.

1. Prostudujte teorii k RW a její možné použití k detekci komunit v komplexních sítích.
2. Přehled algoritmů využívajících RW:
- pro nalezení nového ohodnocení hran v síti.
- pro globální detekci komunit v celé síti.
- pro lokální detekci komunit v síti.
3. Vyberte a popište různě rozsáhlé datové kolekce, nad kterými aplikujete detekci komunit.
3. Implementujte vybrané algoritmy.
4. Porovnejte a analyzujte získané výsledky a vhodně je reprezentujte.

Seznam doporučené odborné literatury:

[1] Alamgir, M., Luxburg, U. von. Multi-agent Random Walks for Local Clustering on Graphs. IEEE International Conference on Data Mining, pp. 18-27 2010
[2] Lovász, L. Random walks on graphs: A survey. Combinatorics Paul Erdos is Eighty 2, pp. 1–46, 1993
[3] Jin, X.-L., Zhu, Z.-Q. & Huang, Z.-L. Search for Directed Networks by Different Random Walk Strategies. Chinese Physics Letters 29, 2012
[4] Pons, P., Latapy, M. Computing communities in large networks using random walks. In Computer and Information Sciences - ISCIS 2005, Springer Berlin Heidelberg, vol. 10, pp. 284-293, 2005

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Pavla Dráždilová, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. května 2014 ……

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Především pak své vedoucí, paní Mgr. Pavle Dráždilové, Ph.D.

Abstrakt

Cílem této práce je popsat vybrané algoritmy pro shlukování dat, které využívají metodu random walk. Na začátku práce si vysvětlíme zákadní pojmy z teorie grafů, které jsou nezbytné pro pochopení dalších částí této práce. Následně si teoreticky popíšeme globální a lokální shlukovací algoritmy. U některých si uvedeme vhodné příklady malého rozsahu. Dále popíšeme, jak byly tyto algoritmy implementovány, s jakými problémy se můžeme setkat a jak tyto problémy řešit. V poslední kapitole si popíšeme testovací data a provedeme sérii experimentů pro ověření funkčnosti algoritmů.

Klíčová slova: komplexní sítě, random walk, shlukování, graf, diplomová práce

Abstract

The aim of this diploma thesis is to describe selected algorithms for data clustering using random walk method. At the beginning, we will explain the bases of the graph theory, which are necessary for the understanding of other parts of this work. Subsequently, a theoretical description of the global and local clustering algorithms takes place. For some algorithms we will show the small scale example. Next, we will describe the implementation of these algorithms, what problems we may encounter and how to solve these problems. In the last chapter, we describe the test data and perform a series of experiments to verify the functionality of the algorithms.

Keywords: complex networks, random walk, clustering, graph, diploma thesis

Seznam použitých zkratek a symbolů

CE	–	Circular Escape
DBLP	–	Databáze literárních spoluautorů
MCL	–	Markov Cluster Algorithm
NS	–	Neighbor Similarity
RW	–	Random Walk
SE	–	Seed Expansion

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Z pojmu *"teorie grafů"* bychom mohli soudit, že se jedná pouze o suchou teorii. Teorie grafů však má mnohá praktická využití a některá z nich jsou velmi zajímavá. Teorie grafů studuje sítě v různých podobách. Takovéto sítě jsou všude kolem nás, aniž bychom si to uvědomovali. Můžeme mít třeba sítě buněk v biologii, nebo sítě počítačové v informatice. Tato práce se zabývá obecnými sítěmi, nejčastěji se však jedná o sítě sociální. Pomocí sociálních sítí studujeme vybranou množinu lidí a vazby mezi nimi. Tématem této práce je hledání komunit (nebo chcete-li shluků) v takovýchto sítích. Komunitou, či shlukem, rozumíme podmnožinu objektů (lidí), které jsou si z nějakého pohledu bližší než s ostatními. Mají tedy něco společného, přičemž se může jednat o různé aspekty. V sociálních sítích to může být například mnoho společných přátel, společné zájmy a podobně. Nalezení správného shluku může být poté v praxi velmi dobře využito. Například když nám síť LinkedIn nabízí další možné kontakty, které by se nám mohly hodit, nebo tyto lidi osobně známe, bylo využito shlukování. Obdobně pracuje také nabízení nových přátel na sociálních sítích Facebook či Twitter.

K nalezení komunit (shluků) se využívá mnoho různých algoritmů, založených na různém principu. V této práci budeme zkoumat algoritmy, které využívají ke shlukování metodu random walk. Jedná se o tzv. náhodnou procházku, kdy například provedeme v síti několik takovýchto náhodných procházek a následně zkoumáme, kudy tato procházka prošla. Je to založeno na principu, že tato náhodná procházka má větší pravděpodobnost zůstat v jednom shluku, než aby jej opustila. Jelikož je zde tedy náhodný faktor, nemusíme dostat pokaždé stejný výsledek.

1.1 Obsah práce

V druhé kapitole se seznámíme s pojmy z teorie grafů, které jsou nezbytné v dalších částech této práce. Po nastudování základů se již můžeme pustit do shlukování. Třetí kapitola se zabývá přehledem algoritmů pro hledání komunit v grafech. Je rozdělena na dvě části: globální shlukovací algoritmy, které hledají shluky v celém grafu, a lokální shlukovací algoritmy, které hledají shluk kolem zvoleného vrcholu. Následuje čtvrtá kapitola - Implementace. Zde si popíšeme, jaké jsou možnosti zpracování grafů v programovacím jazyce C# a jakým způsobem byly implementovány jednotlivé metody pro shlukování. Výsledky shlukování jsou vidět v kapitole páté, která je věnována experimentům a zpracovávaným datům.

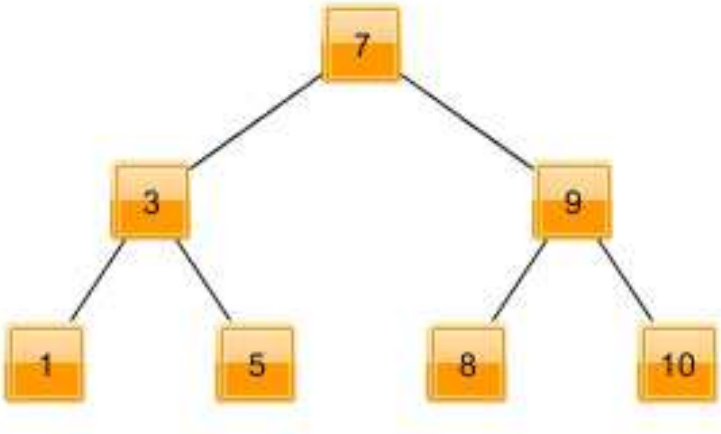
2 Teoretické základy práce

V této kapitole si zavedeme několik pojmů, které budeme dále používat v této práci. Z oblasti teorie grafů si nadefinujeme pojmy jako graf, cesta, sled a další. Dále uvedeme velice důležitý pojem pro tuto práci a to random walk. V českém překladu je to "náhodná procházka". V grafové terminologii se jedná o náhodný sled. Pomocí random walku budeme procházet grafy a hledat v něm komunity. S tím souvisí další pojmy jako stochastická matice nebo markovské procesy. Další podkapitola se věnuje vzdálenostem a podobnostem. Musíme si totiž nadefinovat, jak budeme poznávat dané komunity, neboli jak poznáme, jestli dva vrcholy patří do stejné komunity, či nikoli. Dále se podíváme na shlukovou analýzu, ve které si řekneme, kde a jak se shluková analýza používá, dělení algoritmů apod.

2.1 Teorie grafů

Teorie grafů zkoumá struktury, které známe pod názvem **grafy**. Tyto grafy se skládají z vrcholů a hran. Pomocí grafů můžeme reprezentovat mnoho problémů nejen v matematice, ale i v různých oblastech reálného světa.

Definice 2.1 ***Graf*** *je dvojice* $G = (V, E)$*;, kde* V *je neprázdná množina uzlů (disjunktní s množinou* E*) a* E *je množina hran (disjunktní s množinou* V*).* $\forall e \in E : e = (a, b); a, b \in V$*.* $E \subseteq V \times V$*[1].*



Obrázek 1: Strom

Na začátek si musíme ujasnit, co budeme rozumět grafem. Může to totiž být například graf používaný ve statistice (výsečový, koláčový, sloupcový atd.), nebo třeba grafy funkcí (např. goniometrických apod.). V této práci se však budeme bavit o grafech jako prostředku k popisu reálných problémů, kde můžeme objekty (lidi, křižovatky, města, buňky) popsat jako uzly (či vrcholy) grafu a vztahy (známosti, ulice, cesty, vazby) jsou

reprezentovány hranami grafu. Hrany spojují vrcholy, které mají něco společného. Tyto hrany mohou být ohodnoceny, čímž se vyjádří důležitost daného vztahu (intenzita), nebo jejich vzdálenost. Příklady jednoduchých grafů můžeme vidět na obrázcích 1, 2 a 3. Na obrázku č. 1 je znázorněn graf, který lze označit jako strom, a používá se například v algoritmech pro rychlé vyhledávání. Uzly zde mohou být například čísla a při procházení grafu jen porovnáváme, zda je hledané číslo menší nebo větší než číslo v uzlu, ve kterém se právě nacházíme. Podle tohoto porovnávání pak pokračujeme buďto doleva, nebo doprava. Obrázek č. 2 reprezentuje přátelství mezi konkrétními lidmi. To se často používá v sociálních sítích, kdy např. na Facebooku máme někoho v přátelích a znázorňuje se to právě takovýmto grafem. Na obrázku č. 3 je ohodnocený graf, kde vrcholy zastupují jednotlivá města a ohodnocená hrana mezi nimi pak vyjadřuje vzdálenost mezi těmito městy.

Matematicky se grafy zapisují jako dvojice $G = (V, E)$, kde V označuje konečnou neprázdnou množinu vrcholů a E označuje konečnou množinu hran. Graf může být orientovaný a neorientovaný. V neorientovaném grafu se hrany chápou jako dvouprvková množina $\{u, v\}$ vrcholů $u, v \in V$, kde na pořadí prvků nezáleží. Naopak v orientovaném grafu jsou hrany uspořádané dvojice (u, v) vrcholů u a v. Graf je v tomto případě orientovaný, protože hrana (u, v) je něco jiného než hrana (v, u). V případě ohodnocení hran je třeba graf zapsat jako trojici $G = (V, E, \omega)$, kde V je opět konečná neprázdná množina vrcholů, E je konečná množina hran. ω je funkce $\omega : E \to \mathbb{R}^+$, která ke každé hraně přiřadí číselnou hodnotu (ve většině případů $\mathbb{R}^+$, možno i $\mathbb{R}$).

Definice 2.2 ***Sled*** *v grafu je taková posloupnost vrcholů, kdy mezi každými po sobě jdoucími vrcholy existuje hrana [1].*

Definice 2.3 ***Tah*** *je sledem, ve kterém se nevyskytuje žádná hrana více než jednou. Tedy pokud tah obsahuje za sebou vrcholy uv, pak už dále neobsahuje uv, ani vu. Rozlišujeme uzavřený tah, který začíná i končí ve stejném vrcholu, a otevřený tah, kde je počáteční uzel různý od koncového uzlu. Pokud vede tah přes všechny vrcholy, říká se mu eulerovský [1].*

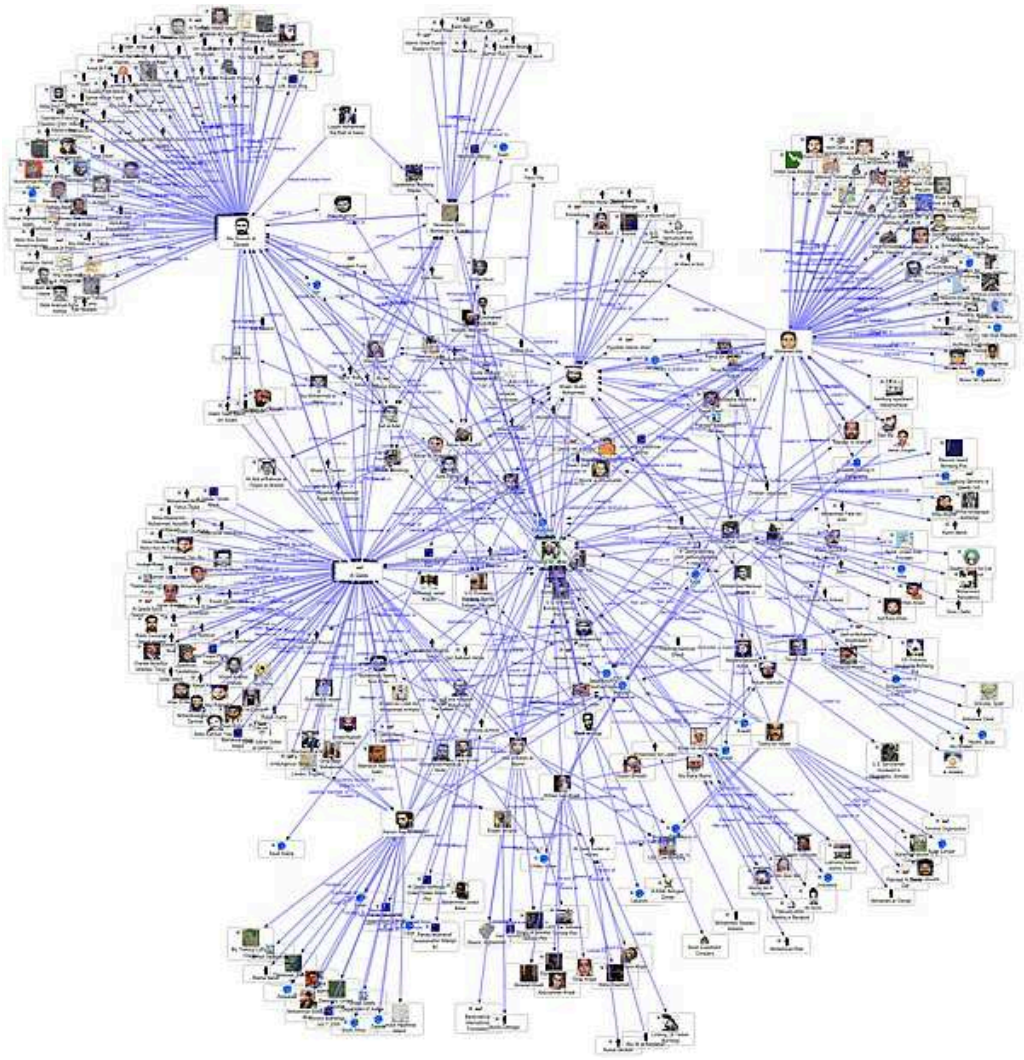
Definice 2.4 ***Cesta*** *je tahem, ve kterém se neopakují vrcholy. Tedy každý vrchol je použit jen jednou. Existuje-li mezi vrcholy sled, pak mezi nimi existuje i cesta [1].*

Definice 2.5 ***Souvislá komponenta*** *je podgrafem původního grafu, v němž platí, že pro každé dva vrcholy x, y existuje alespoň jedna cesta z x do y [1].*

Definice 2.6 ***Stupeň vrcholu*** *označuje počet hran, které do daného vrcholu zasahují. Stupeň vrcholu u se značí $deg(u)$, kde $deg(u) = |\{e \in E | u \in e\}|$ [1].*

Definice 2.7 ***Nesouvislý graf*** *je grafem, v němž platí, že pro alespoň dva vrcholy x, y neexistuje cesta z x do y.*

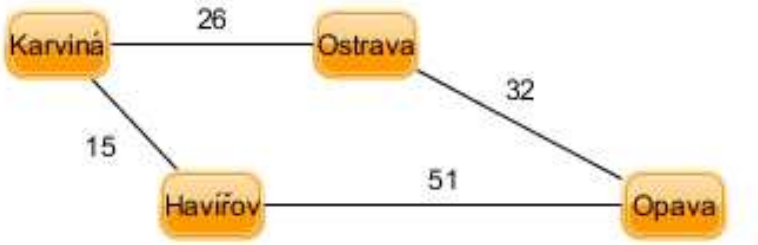
Definice 2.8 ***Komunita*** *je podgraf, který má mnoho hran uvnitř komunity a málo hran vedoucích do jiných komunit.*



Obrázek 2: Graf sociální sítě [18]

2.2 Komplexní síť

Algoritmy, které jsou v této práci představeny, mohou pracovat s jakoukoliv strukturou grafů. Prakticky se však využívají především pro zpracování komplexních sítí. Komplexní sítě mají své vlastnosti a specifika. Tyto vlastnosti se nevyskytují v klasických grafech, jako třeba tzv. mesh graf (který si můžeme představit třeba jako čtvercovou síť, jejíž podgrafy jsou opět čtverce), nebo náhodný graf. Komplexní síť často reprezentuje reálné problémy. Používá se v mnoha oborech jako biologie, informatika, nebo sociologie. V komplexní síti se často vyskytují komunity, které jsou středem našeho zájmu.



Obrázek 3: Graf vzdálenosti měst

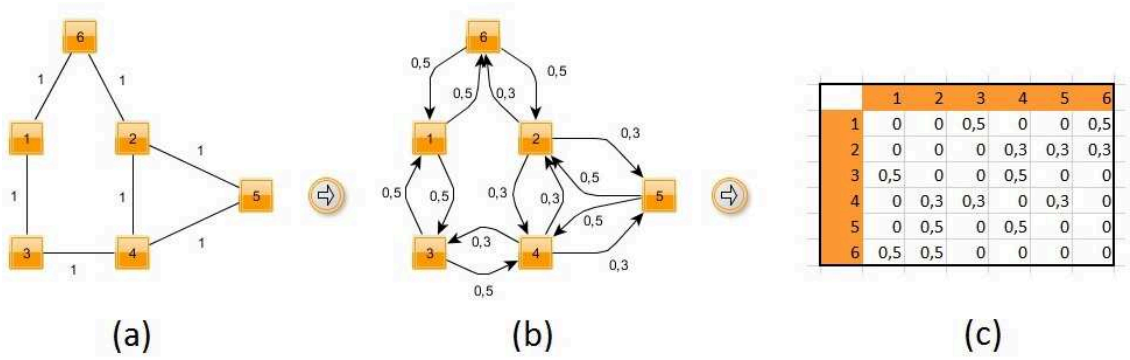
2.3 Random walk

Česky „náhodná procházka“, v textu se však budu držet původního anglického názvu random walk. Jde o matematickou formulaci sledu, který se skládá z náhodných kroků. Pro rychlé pochopení uveďme několik příkladů. Může se jednat například o sled pohybů molekuly v kapalině či plynu, nebo sled přesunů zvířete na louce. Dobrým příkladem může být také finanční stav hráče v herně (tzv. gamblera). Toto vše může být modelováno jako random walk, i když ne vždy se jedná o skutečně náhodné výběry dalšího kroku. Termín random walk poprvé použil Karl Pearson v roce 1905. Od té doby se využívá v mnoha oborech jako ekonomie, ekologie, informační technologie, chemie, nebo třeba biologie. Předmětem zájmu jsou různé typy random walku. Často jsou reprezentovány pomocí markovských procesů. Některé složitější jsou pak vyjadřovány pomocí grafů, rovin, nebo ve vyšších dimenzích, pokud je potřeba zaznamenat více proměnných. Někdy se také počítá s časem, kdy náhodné výběry dalšího kroku mohou být provedeny v náhodných časech. Nejjednodušším příkladem je jednorozměrný random walk. Představme si řadu celých čísel a začněme v nule. Můžeme třeba házet poctivou mincí a pokud padne hlava, posuneme se doleva. Pokud padne orel, posuneme se doprava. Každý krok tedy bude buďto +1 nebo -1, a to se stejnou pravděpodobností (tj. 0,5). Po pěti hodech pak může být výsledek jedno z čísel: -5,-3,-1,1,3,5.

Ve složitějším grafu ve více dimenzích už to tak jednoduché samozřejmě nebude. My budeme používat ohodnocený graf, který si musíme upravit na graf pro RW. Taková transformace je zobrazena na obrázku 4. Ohodnocení hran vyjadřuje pravděpodobnost přechodu z v_i do v_j a vypočítá se jako podíl hodnoty dané hrany a součtu hodnot všech hran, incidentních s daným vrcholem. Musí platit, že $\sum_{\forall j} p_{ij} = 1$. Jelikož každá hrana sousedí se dvěma vrcholy, dostaneme místo každé hrany dvě nové, orientované hrany. Transformovaný graf můžeme zapsat pomocí řádkově stochastické matice (obr. 4(c)), která je popsána níže.

2.3.1 Stochastická matice

Stochastická matice má pouze nezáporné prvky a dále se rozděluje na sloupcovou a řádkovou. Nás bude zajímat řádkově stochastická matice, která má také pouze nezáporné



	1	2	3	4	5	6
1	0	0	0,5	0	0	0,5
2	0	0	0	0,3	0,3	0,3
3	0,5	0	0	0,5	0	0
4	0	0,3	0,3	0	0,3	0
5	0	0,5	0	0,5	0	0
6	0,5	0,5	0	0	0	0

Obrázek 4: Transformace grafu na graf random walk a následně na stochastickou matici

prvky a navíc je součet každého řádku roven jedné. Příklad řádkově stochastické matice můžeme vidět na obrázku 4(c). Matematická definice může být zapsána jako [9]:

$$\forall i \forall j \; p_{i,j} = \frac{w_{i,j}}{\sum_{k=1}^{n} w_{i,k}}$$

kdy platí, že:

$$p_{ij} \geq 0, \qquad \sum_j p_{ij} = 1, \qquad i,j = 1,\ldots,n$$

neboli

$$P \geq 0, \qquad Pe = e, \qquad e = (1,\ldots,1)^\tau$$

2.3.2 Markovské řetězce

Definice 2.9 ***Markovské řetězce:***

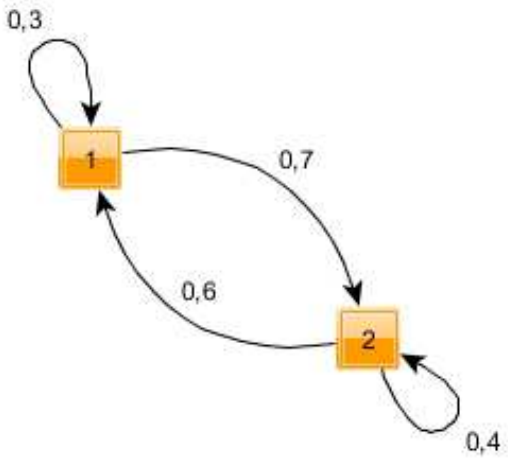
1. $p_{ij}^{(m)}$ *je podmíněná pravděpodobnost přechodu daného systému při m-tém kroku ze stavu i do stavu j*
2. $p(s_i \to s_j) = p_{ij}$
3. *matice přechodu markovského řetězce je sestavena z podmíněných pravděpodobností přechodů:*

$$P = \begin{pmatrix} p_{1,1}^{(m)} & p_{1,2}^{(m)} & \cdots & p_{1,n}^{(m)} \\ p_{2,1}^{(m)} & p_{2,2}^{(m)} & \cdots & p_{2,n}^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1}^{(m)} & p_{m,2}^{(m)} & \cdots & p_{m,n}^{(m)} \end{pmatrix}$$

4. *markovský řetězec je homogenní, když podmíněné pravděpodobnosti přechodu* $p_{ij}^{(m)}$ *nezávisejí na m, tj. pro všechna i,j platí:*

$$p_{ij}^{(m)} = p_{ij}, P^{(m)} = P = \{p_{ij}\}$$

Markovským řetězcem popisujeme přechody z jednoho stavu do druhého. Tyto přechody pak mají důležitou vlastnost: Každý přechod je závislý pouze na současném stavu, tudíž je nezávislý na stavech předchozích. Neexistuje zde tedy žádná paměť. Díky této vlastnosti můžeme takovýto proces jednoduše znázornit na obrázku (v grafu). Příklad takového grafu je na obrázku č. 5. Markovské řetězce můžeme reprezentovat pomocí stochastické matice, jak je vidět z transformace na obr. 4.



Obrázek 5: Markovův řetězec, reprezentován ohodnoceným a orientovaným grafem.

2.4 Vzdálenosti a podobnosti

Vzdálenosti a podobnosti v grafu můžeme měřit několika způsoby. V případě neohodnoceného grafu si je musíme vypočítat. V případě ohodnoceného grafu můžeme danou hodnotu prohlásit za vzdálenost či podobnost dvou vrcholů, incidentních s příslušnou hranou. Tyto pojmy si musíme zadefinovat z důvodu, že pro pozdější výpočty vždy potřebujeme podobnost. Na vstupních datech však může být vzdálenost (např. pro města na mapě) a proto ji musíme převést na podobnost.

2.4.1 Euklidovská metrika

Jedná se o základní způsob měření vzdálenosti dvou objektů x a y při hledání shluků. Tyto objekty jsou reprezentovány pomocí vektorů ($x, y \in \mathbb{R}^n$). Princip je podobný jako v geometrii, kdy počítáme délku přepony pravoúhlého trojúhelníku pomocí Pythagorovy věty [11].

$$d_E = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2.4.2 Vzdálenost

Abychom mohli vůbec měřit vzdálenost objektů, musí tyto objekty splňovat několik pravidel. Objekty si označíme O_i, $i = 1, \ldots, n$, vzdálenost pak bude d.

Definice 2.10 *Vzdálenost dvou objektů musí splňovat následující podmínky [11]:*

- *Nezápornost:* $d(O_{i,j}) \geq 0 \ \forall i, j$ *- vzdálenost dvou objektů nemůže být záporná*
- *Identita:* $O_i = O_j \rightarrow d(O_i, O_j) = 0$ *- vzdálenost objektu od sebe samého je nulová*
- *Symetrie:* $d(O_i, O_j) = d(O_j, O_i)$ *- vzdálenost z obou stran je stejná*
- *Trojúhelníková nerovnost:* $d(O_i, O_j) \leq d(O_i, O_k) + d(O_j, O_k)$ *- vlastnost vychází ze základního pravidla konstrukce trojúhelníku, tedy že součet kterýchkoli dvou stran musí být větší než strana třetí*

2.4.3 Podobnost

Podobnost můžeme vypočítat pomocí vzdálenosti, kdy využijeme těchto vztahů (s značíme podobnost z anglického similarity, d značíme vzdálenost z anglického distance):

$$d \in < 0, 1 > \rightarrow s = 1 - d, \text{ nebo } s = e^{-d}$$

a naopak:

$$s \in (0, 1 > \rightarrow d = \frac{1}{s} - 1, \text{ nebo } d = 1 - s$$

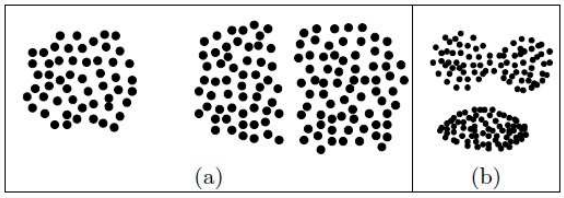
Nulová podobnost nám říká, že mezi vrcholy není hrana. Jak vzdálenost tak podobnost musíme ze vstupních dat normovat na interval $< 0, 1 >$. Normování provedeme tak, že vydělíme všechny prvky množiny čísel největším prvkem této množiny. Nejvyšší hodnota bude tedy 1.

2.5 Shluková analýza

Termín byl poprvé použit Robertem Tryonem již v roce 1939. Shluková analýza se zabývá metodami a algoritmy, které sdružují podobné objekty do shluků. Jde tedy o vytvoření skupin objektů (**shluků**) s maximálně podobnými vlastnostmi. Pokud budou tyto objekty reprezentovat lidi, hovoříme o **komunitách**. Mezi těmito skupinami objektů jsou podobnosti minimální a naopak vzdálenosti jsou velké. Mezi těmito objekty tedy můžeme měřit podobnost, či vzdálenost [10].

Problém shlukování v grafu je dobře popsán také v [2]. Pánové Harel a Koren zde vyjadřují myšlenku, že není vždy jasné, na jaké komunity se má původní graf rozdělit. Kvalitní algoritmus by v jejich případě měl vrcholy rozdělovat do určitého počtu přirozených komunit. Reálně však většina algoritmů vyžaduje toto číslo (počet komunit) jako vstup. To však znamená, že algoritmus může přirozené komunity oddělit, sloučit k sobě, či vytvořit nové (nepřirozené).

Jak tedy poznat přirozený shluk v grafu? Ani toto není jasně dané. Uveďme si příklad z výše uvedené literatury. Na obrázku 6(a) vidíme na první pohled tři oddělné komunity. Někdo by však mohl namítat, že dvě komunity na pravé straně jsou u sebe dost blízko a lze je tedy považovat za jednu komunitu. Ani na obrázku 6(b) není rozdělení jasné. Dolní část vrcholů bude samostatná komunita automaticky, na tom se většinou shodneme. Ale horní část už tak jasná není. Jedná se o jednu komunitu, nebo je oddělení dostatečné, abychom mohli říct, že se jedná o dvě samostatné komunity? Toto rozdělení ovlivníme pomocí hierarchických shlukovacích algoritmů, které nám poskytují parametr pro určení míry rozdělení.



Obrázek 6: Možnosti rozdělení vrcholů do shluků [2]

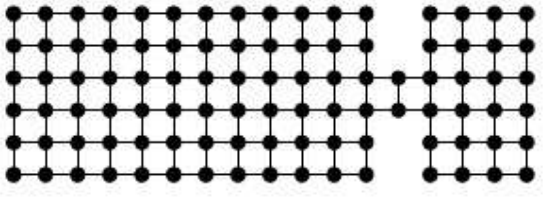
Byly navrženy různé algoritmy pro měření kvality shluků. Nicméně neexistuje funkce s polynomiální časovou složitostí, která by optimálně zachytila pojem "dobrý shluk". Vždy můžeme najít případ, pro který tato funkce selže. Slouží tedy pouze jako intuice pro rozlišování dobrých shluků od špatných. Některé možnosti měření kvality shluků si ukážeme dále.

2.5.1 Základní dělení algoritmů

Metody pro shlukovací analýzu se dělí na dvě skupiny [2]. Jedná se o hierarchické metody a nehierarchické, v některých publikacích též zvané dělící metody (z anglického partitional).

Nehierarchické algoritmy optimalizují dělení množiny vstupních vrcholů do skupin podle určitého kritéria. Obvykle je tímto kritériem minimalizace vzdálenosti všech vrcholů ve shluku od prostředního vrcholu (centra shluku). Průnik těchto skupin (množin, shluků) je vždy prázdný, tzn. jsou navzájem disjunktní. Tato metoda funguje dobře na izolovaných a kompaktních datech. Mezi výhody této metody patří jednoduché odstranění velmi vzdálených vrchoů, které do přirozeného shluku nepatří. Další výhodou je rychlost samotného algoritmu. Nevýhodou je potom vytváření zaoblených shluků stejné velikosti, které by přirozeným způsobem nevznikly. Tuto myšlenku lze demonstrovat pomocí obrázku č. 7. Přirozeným způsobem bychom vrcholy na obrázku rozdělili do dvou skupin (shluků). Větší část nalevo a menší část napravo. Nehierarchické metody však mají tendenci přiřadit několik vrcholů z levé části k té pravé. To právě z důvodu zkrácení vzdáleností vrcholů od center shluků. Jako příklad konkrétního nehierarchického algoritmu si můžeme uvést například algoritmus k-means.

Hierarchické algoritmy jsou, na rozdíl od nehierarchických, systémem podmnožin. Průnikem dvou podmnožin může opět být prázdná množina, ale průnikem může být



Obrázek 7: Do jakých komunit se rozdělí tato množina? [2]

také jedna z těchto množin. Pokud se tak stane alespoň jednou, jedná se o hierarchický systém. Postup může být dvojího typu. Buďto vezmeme celou množinu jako celek a ten postupně dělíme na menší části (divizní přístup), nebo začneme s n shluky na n vrcholech o velikosti 1. Takže každý vrchol je samostatným shlukem. Postupujeme tak, že v každém dalším kroku sloučíme nejvíce podobné páry shluků do jednoho (aglomerativní přístup). Nejrozšířenější varianty aglomerativního přístupu jsou single-linkage a complete-linkage. V algoritmu single-link se porovnávají nejvíce podobné vrcholy (z každého shluku jeden), zatímco v complete-link algoritmu se porovnávají nejméně podobné vrcholy (opět z každého shluku jeden). Complete-link algoritmus má tendenci rozbít velký (ale přirozený) shluk do dvou (nepřirozených). U metody single-link pak je problém v tom, že jej mohou "oklamat"vzdálené vrcholy shluku, spojené třeba jen jediným řetězcem hran, tzv. řetězový efekt.

2.5.2 Měření kvality shluků

Různé shlukovací algoritmy naleznou různé shluky. My ale potřebujeme ověřit, zda se vůbec jedná o shluk a jakou má kvalitu. Pro malá data můžeme výsledek ověřit pouhým okem. V rozsáhlých datech však tuto možnost nemáme a nemůžeme jednoduše ověřit výsledky. Dalším důvodem pro měření kvality shluku v grafu může být přizpůsobení shlukovacího algoritmu. Některé mají dokonce kvalitativní parametry jako svůj vstup a podle něj vyhledávají shluky. Kvalitu shluku v grafu můžeme měřit různými způsoby. My zde zmíníme jen některé z nich.

Jednou z nejznámějších metod pro měření kvality nalezených shluků v grafu je **modularita** [5]. Modularitou vyjadřujeme, že nalezený shluk obsahuje více vnitřních hran, než bylo předpokládáno a naopak méně hran spojujících shluky (tj. hrana která má pouze jeden koncový vrchol ve shluku) v porovnání s náhodným grafem s podobnou strukturou. Výpočet hodnoty modularity Q je dán rovnicí: $Q = Tr(e) - ||e^2||$, kde e je symetrická matice, jejíchž elementy e_{ij} jsou hrany v síti, které spojují komunity i a j. $Tr(e)$ je tzv. stopa matice e, neboli součet všech elementů na její hlavní diagonále.

Je však více možností, kterak spočítat modularitu shluků. Zmíníme zde ještě výpočet, který jsme později použili v implementaci. Vzorec (1) je převzat z [7]:

$$(1) \quad Q(C) = \sum_{c \in C} [\frac{|E(c)|}{m} - (\frac{\sum_{v \in C} deg(v)}{2m})^2],$$

kde zlomek $\frac{|E(c)|}{m}$ udává poměr všech ohodnocení hran daného shluku ku ohodnocení hran celého grafu shluku. Této metodě se také říká coverage a je popsána dále. Druhý výraz $(\frac{\sum_{v \in C} deg(v)}{2m})^2$ je poměrem součtu stupňů všech vrcholů dané komunity a dvojnásobku celkového ohodnocení hran. Toto násobení se děje proto, že započítáváme každou hranu dvakrát a tímto tuto skutečnost eliminujeme.

Výsledkem této rovnice, a tedy modularity Q, je hodnota z intervalu $(-1/2, 1)$ Hodnota blížící se k 1 znamená velice silnou komunitní charakteristiku. V některých případech nám může vyjít záporná hodnota. A to v případě, že máme shluk, ve kterém je pouze jeden vrchol a tudíž žádné vnitřní hrany.

Další varianta měření kvality shluků v grafu se nazývá **konduktance** [5] (přeloženo ze slova conductance - pojem pochází z elektrotechniky, též vodivost). Graf se rozdělí do podgrafů a měří se počet odstraněných hran. Dále se měří váhy hran ve vzniklých podgrafech. Představme si rozdělení grafu G na disjunktní shluky $C_1, C_2 \dots C_k$. Konduktance pro všechny shluky C_i je počítá jako:

$$(2) \quad C_i = \frac{\sum_{u \in C_i} \sum_{v \notin C_i} w(\{u,v\})}{min(a(C_i), a(\overline{C_i}))},$$

kde $a(C_i) = \sum_{u \in C_i} \sum_{v \in V} w(u, v)$ je suma všech ohodnocení hran, které mají alespoň jeden vrchol ve shluku C_i. Pokud chceme najít k shluků, musíme provést $k - 1$ rozdělení hran. Konduktance pro celý původní graf můžeme poté prezentovat jako průměr pro všechny $(k - 1)$ řezy: $avg(C_i), \forall C_i \subseteq V$.

Další možnou metrikou je tzv. hustota z anglického **density**. Ta se vypočítá jako poměr počtu hran v daném shluku, vzhledem k maximálnímu možnému počtu hran. Pokud má graf maximální možný počet hran, jedná se o graf úplný. Pokud si označíme počet vrcholů jako n, pak se počet hran v úplném grafu spočítá jako $\frac{n*(n-1)}{2}$.

$$(3) \quad density(C) = \frac{1}{k} \sum_{i=1}^{k} \frac{|C_i|}{|complete\ graph\ for\ C_i|}.$$

Pro lokální komunity budeme hustotu počítat jako počet hran dané komunity ku maximálnímu možnému počtu hran pro danou komunitu:

$$(4) \quad density(C)_{local} = \frac{|C|}{|complete\ graph\ for\ C|}.$$

Ještě popíšeme výše zmíněnou možnost měření kvality shluku, a to **coverage**, neboli pokrytí [5]. Jedná se o poměr součtu ohodnocení hran v daném shluku a součtu ohodnocení hran v celém grafu G. Matematicky tedy zapsáno jako:

$$(5) \quad coverage(C) = \sum_{i}^{k} \frac{w(C_i)}{w(G)}, \text{ kde } w(C_i) = \sum w(v_x, v_y); \forall v_x, v_y \in C_i.$$

Opět dostaneme hodnoty z intervalu $(0, 1)$, kde větší číslo znamená, že je zde více vnitřních hran, než hran směřujících ven ze shluku.

2.5.3 Využití shlukování

Shlukovací analýza má více možných použití, než by se mohlo na první seznámení s tímto oborem zdát. Používá se v mnoha oblastech, proto zmiňme jen některé z nich. V business sféře se shlukování používá k oddělení skupin zákazníků a tedy k lepšímu přehledu trhu. Firma pak může nabízet ty správné produkty těm správným potencionálním zákazníkům. Důležitým oborem je také lékařství. Shlukování zde slouží pro kategorizaci genů, z čehož vyplývá další možnost využití shlukování - v genetice. Při dolování dat může shluková analýza prozradit další důležité informace o získaných datech. Své využití najde shluková analýza také v dalších oborech jako statistika, biologie, nebo ekonomie. Velice zajímavé téma je pro nás využití shlukování v sociálních sítích. Zde už se v praxi s úspěchem používá. Například jde o navrhování nových kontaktů na síti Facebook. Představme si graf takové sociální sítě. Hustě propojené místo se nazývá komunita. Pokud jsme v takové komunitě a nemáme v přátelích někoho ze stejné komunity (tj. neexistuje s ním hrana v grafu), Facebook nám jej nabídne jako potencionálního přítele.

3 Analýza shluků pomocí metody random walk

V předchozí kapitole jsme si vysvětlili, co chápeme pod pojmem random walk. Nyní si ukážeme, jak jej můžeme využít ve shlukové analýze. Random walk použijeme pro vyhledávání shluků díky jeho známé vlastnosti. Představme si dva shluky A a B v grafu $G = (V, E)$. Vrcholy v těchto shlucích jsou hustě propojeny hranami. Mezi shluky je naopak počet hran velice nízký. Začneme tím, že si vybereme libovolný vrchol i ze shluku A. Z tohoto vrcholu vyjdeme náhodným směrem do vrcholu j. Je více pravděpodobné, že zůstaneme ve stejném shluku (A), než že bychom použili tzv. separační hranu a shluk opustili (tj. dostali bychom se do shluku B).

Této vlastnosti využívají mnohé algoritmy, jako například MCL - Markov cluster algorithm, který si popíšeme níže.

3.1 Globální shlukovací algoritmy

Globální shlukovací algoritmy pracují s celým vstupním grafem a hledají tedy shluky v celém grafu. Lokální shlukovací algoritmy naopak pracují jen okolo zvoleného vrcholu, který nazýváme centrálním vrcholem. Tento vrchol je vstupem shlukovacího algoritmu.

3.1.1 Separace pomocí nového ohodnocení hran

Jednou z variant, jak nalézt shluky v grafu je separace pomocí nového ohodnocení hran, jako je to popsáno v [2]. Jak již název napovídá, využívá se ohodnocení hran. Pokud máme neohodnocený graf, dáme všem hranám hodnotu 1. Princip je následující: postupně procházíme všechny vrcholy v grafu a porovnáváme dvojice. Měříme míru intiminty, tedy jak moc jsou si podobné. Pokud jsou si blízko, tedy měly by patřit do jednoho shluku, zvýšíme hodnocení hrany mezi těmito vrcholy. Pokud jsou si vzdálené (tj. patří do různých shluků), snížíme ohodnocení dané hrany. Toto opakujeme do té doby, než můžeme jasně vyhodnotit výsledky. Tj. máme buďto hodnoty blížící se nule, nebo naopak hodnoty vysoké.

Nyní si představíme konkrétní metody pro nové ohodnocení hran. Jde o metody Neighborhood Similarity a Circular Escape.

3.1.1.1 Neighborhood similarity

Jedná se o algoritmus využívající separaci hran, o které je předchozí odstavec. Využívá se tedy nového ohodnocení hran. Hrany s malou hodnotou odstraníme, čímž nám vzniknou nové shluky. Problémem může být zvolení správné hraniční hodnoty (tzv. treshold), kterou určíme nevýznamné hrany.

Definice 3.1 *$P^k_{visit}(i) \in \mathbb{R}$ je vektor, jehož j-tá komponenta je pravděpodobnost, že náhodná procházka začínající ve vrcholu i navštíví vrchol j v jeho k-tém kroku. Neboli $P^k_{visit}(i) \in \mathbb{R}$ je i-tý řádek matice P^k, tj. k-tá mocnina matice P [2].*

Vektor $P^k_{visit}(i)$, uvedený v definici 3.1, nám poskytne míru intimity mezi vrcholem i a všemi jeho sousedy. Jde v podstatě o zobecnění konceptu ohodnocení hran, kdy $P^1_{visit}(i)$ jsou hodnoty hran k sousedům od vrcholu i. Hodnoty $P^k_{visit}(i)$ však pro nás zajímavé nebudou a nejsou ani vhodné pro větší hodnoty k. Zavedeme si proto značení $P^{\leq k}_{visit}(i)$, které definujeme jako $\sum_{i=1}^{k} P^i_{visit}(v)$.

Nyní si dosadíme malé číslo do proměnné k. Většinou to budou hodnoty $k = 2$, nebo $k = 3$ a porovnáme hodnoty $P^{\leq k}_{visit}(u)$ a $P^{\leq k}_{visit}(v)$. Čím menší bude rozdíl těchto hodnot, tím větší je mezi vrcholy u a v podobnost (intimita) [2].

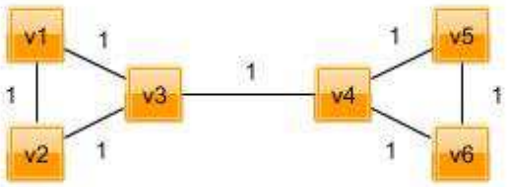
Samotnou hodnotu podobnosti můžeme vypočítat různými způsoby. Uvedeme vzorce z [2]. Jeden způsob výpočtu podobnosti je dle vzorce:

$$f^k(x, y) = exp(2k - ||x - y||L_1) - 1. \quad (5)$$

Druhý způsob je pak podle vzorce:

$$\cos(x, y) = \frac{(x,y)}{\sqrt{(x,x)}.\sqrt{(y,y)}} \quad (6)$$

Příklad 3.1
Pro lepší pochopení si nyní uveďme příklad. Na obrázku 8 máme graf, který se na první pohled skládá ze dvou shluků. Jeden v levé části a druhý v pravé části obrázku. Graf je původně neohodnocený, proto budeme uvažovat hodnotu každé hrany 1.



Obrázek 8: Souvislý, neorientovaný graf.

Vytvoříme si matici sousednosti A, a pravděpodobnostní matici P:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0.33 & 0.33 & 0 & 0.33 & 0 & 0 \\ 0 & 0 & 0.33 & 0 & 0.33 & 0.33 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 \end{pmatrix}$$

Nyní si spočítáme matici všech vektorů $P^{\leq k}_{visit}(i)$ dle definice 3.1 pro konstatntu $k = 2$:

$$\sum_{i=1}^{2} P_{visit}^{i} = P + P^2 = \begin{pmatrix} 0.415 & 0.665 & 0.750 & 0.165 & 0.000 & 0.000 \\ 0.665 & 0.415 & 0.750 & 0.165 & 0.000 & 0.000 \\ 0.495 & 0.495 & 0.439 & 0.330 & 0.109 & 0.109 \\ 0.109 & 0.109 & 0.330 & 0.439 & 0.495 & 0.495 \\ 0.000 & 0.000 & 0.165 & 0.750 & 0.415 & 0.665 \\ 0.000 & 0.000 & 0.165 & 0.750 & 0.665 & 0.415 \end{pmatrix}$$

Nyní se podíváme na konkrétní vektory a vypočítáme pro ně jejich intimitu. To nám pomůže při určování, které vrcholy patří do jednoho shluku a které shluky naopak oddělují. Jako první si vezmeme vrcholy z jednoho shluku $v1$ a $v2$.

$$P_{visit}^{\leq 2}(v1) = \begin{pmatrix} 0.415 & 0.665 & 0.750 & 0.165 & 0.000 & 0.000 \end{pmatrix}$$
$$P_{visit}^{\leq 2}(v2) = \begin{pmatrix} 0.665 & 0.415 & 0.750 & 0.165 & 0.000 & 0.000 \end{pmatrix}$$

Nyní si spočítáme samotnou hodnotu podobnosti pro vrcholy $v1$ a $v2$ pomocí vzorce uvedeného výše:

$$\cos(P_{visit}^{\leq 2}(v1), P_{visit}^{\leq 2}(v2)) = \frac{(P_{visit}^{\leq 2}(v1) P_{visit}^{\leq 2}(v2))}{\sqrt{(P_{visit}^{\leq 2}(v1) P_{visit}^{\leq 2}(v1))} . \sqrt{(P_{visit}^{\leq 2}(v2) P_{visit}^{\leq 2}(v2))}} = \frac{1.142}{1.097 * 1.097} = 0.949$$

Pro porovnání si nyní vezmeme vrcholy, které oddělují shluky. Tedy vrcholy $v3$ a $v4$.

$$P_{visit}^{\leq 2}(v3) = \begin{pmatrix} 0.495 & 0.495 & 0.439 & 0.330 & 0.109 & 0.109 \end{pmatrix}$$
$$P_{visit}^{\leq 2}(v4) = \begin{pmatrix} 0.109 & 0.109 & 0.330 & 0.439 & 0.495 & 0.495 \end{pmatrix}$$

Stejným způsobem jako v předchozím případě vypočítáme hodnotu podobnosti pro vrcholy $v3$ a $v4$:

$$\cos(P_{visit}^{\leq 2}(v3), P_{visit}^{\leq 2}(v4)) = \frac{(P_{visit}^{\leq 2}(v3) P_{visit}^{\leq 2}(v4))}{\sqrt{(P_{visit}^{\leq 2}(v3) P_{visit}^{\leq 2}(v3))} . \sqrt{(P_{visit}^{\leq 2}(v4) P_{visit}^{\leq 2}(v4))}} = \frac{0.506}{0.903 * 0.903} = 0.621$$

Z výsledků je patrné, že větší hodnota u vrcholů $v1$ a $v2$ signalizuje větší pravděpodobnost umístění těchto vrcholů do jednoho shluku. Naopak menší hodnota u vrcholů $v3$ a $v4$ ukazuje na možnost, že se jedná o separační hranu. Pokud provedeme tento algoritmus vícekrát (iterativně), budou se tyto hodnoty dále vzdalovat. ■

Ještě si nadefinujme separační algoritmus a podobnostní rovnici. Definice 3.2 je převzata z [2].

Definice 3.2 *Nechť $G(V, E, w_s)$ je ohodnocený graf a k je nějaká malá konstanta. Rozdělení grafu G pomocí Neighborhood Similarity (podobnost sousedů), označenou jako $NS(G)$, je definováno:*

$NS(G) = G_s(V, E, w_s)$,

kde $\forall(u,v) \in E, w_s(u,v) = sim^k(P^{\leq k}_{visit}(v), P^{\leq k}_{visit}(u))$,
kde $sim^k(x,y)$ *je podobnostní funkce vyjádřená v odstavci Neighborhood similarity výše.*

Algoritmus výpočtu NS:
Vstup: Graf G reprezentovaný řádkově stochastickou maticí.
Výstup: Nově ohodnocený graf G.

1. Pro všechny hrany $e \in E$ z grafu G spočítej vektory $P^{\leq k}_{visit}(v)$ a $P^{\leq k}_{visit}(u)$, kde u, v jsou incidentní s hranou e.
2. Spočítej podobnost mezi vektory $P^{\leq k}_{visit}(v)$ a $P^{\leq k}_{visit}(u)$. Hodnotu ulož do pomocné struktury k příslušné hraně ($h(e)$). Pokračuj bodem 3.
3. Pro všechny hrany $e \in E$ z grafu G aktualizuj ohodnocení hran pomocí nově spočtených hodnot z bodu 2. Tj. $w(e) := h(e)$.

3.1.1.2 Circular Escape

Alternativa k metodě Neihborhood Similarity je tzv. Circular Escape. Také touto metodou měříme míru intimity mezi jednotlivými vrcholy grafu. Jde o pravděpodobnost, že náhodná procházka (random walk) začínající ve vrcholu v navštíví také vrchol u právě jednou předtím, než se vrátí zpět do vrcholu v. Tato notace je symetrická, což bude vidět v definici této pravděpodobnosti níže. Pokud jsou vrcholy u a v v různých shlucích, pravděpodobnost bude malá. RW se s větší pravděpodobností dříve vrátí do startovního vrcholu, než navštíví cílový. Naopak pokud jsou vrcholy u a v ve stejném shluku, bude tato pravděpodobnost velká. Pravděpodobnost budeme značit jako: $P_{escape}(v,u).P_{escape}(u,v)$.

Metoda Circular Escape vždy počítá s podgrafem původního grafu G. Tento podgraf zahrnuje blízké okolí vrcholů u a v, mezi nimiž počítáme podobnost (intimitu). Tento podgraf je zaznamenán v definici 3.3, která je převzata z [2].

Definice 3.3 *Nechť* $G(V,E,w)$ *je graf a* k *je nějaká malá konstanta.* $P^{(k)}_{escape}(v,u)$ *budiž označení pravděpodobnosti* $P_{escape}(v,u)$*, která je ale počítána pomocí random walku na podgrafu* $G(V^k, (\{u,v\}))$*, místo původního grafu* G*. Pravděpodobnost Circular Escape je poté definována jako:*

$$CE^k(v,u) = P^{(k)}_{escape}(v,u).P^{(k)}_{escape}(u,v)$$

Nyní si můžeme nadefinovat separaci pomocí Circular Escape [2]:

Definice 3.4 *Nechť* $G(V,E,w)$ *je ohodnocený graf a* k *je nějaká malá konstanta. Separace grafu* G *pomocí Circular Escape, označenou jako* $CE(G)$ *nadefinujeme jako:*

$CE(G) = G_s(V,E,w_s)$
kde $\forall(u,v) \in E, w_s(u,v) = CE^k(v,u)$

Algoritmus výpočtu CE:
Vstup: Graf G reprezentovaný řádkově stochastickou maticí.
Výstup: Nově ohodnocený graf G.

1. Pro všechny hrany $e \in E$ z grafu G proveď krok 2.
2. Spočítej soustavy rovnic pro výpočet pravděpodobnosti $P^{(k)}_{escape}(v,u)$ a $P^{(k)}_{escape}(u,v)$. Pokračuj bodem 3.
3. Z vyřešené soustavy z bodu 2 vypočti násobek vektorů $P^{(k)}_{escape}(v,u)$ a $P^{(k)}_{escape}(u,v)$. Pokračuj bodem 4.
4. Aktualizuj ohodnocení hran pomocí nově spočtených hodnot z bodu 4.

3.1.2 MCL - Markov cluster algorithm

Algoritmus byl poprvné představen v disertační práci Stijina van Dongena - Graph clustering by flow simulation [15]. Tato metoda využívá vlastnosti, že random walk prochází jednodušeji hustou oblastí v grafu, než v řídkých hranicích shluků. V dlouhodobém horizontu (dlouhý tah) tento efekt však mizí. Standardně se v této metodě používá sloupcově stochastická matice (dosud jsme používali řádkově stochastickou matici).

Každá iterace algoritmu MCL se skládá ze dvou kroků. První krok se nazývá **rozšíření**. V tomto kroku se matice umocní na konstantu k (rozšiřovací parametr), většinou $k = 2$. Tímto vznikne matice M_{ij} která nám udává pravděpodobnosti, že random walk, začínající ve vrcholu j dosáhne vrcholu i v k krocích. Určujeme tím také možnost připojení různě vzdálených bloků vrcholů. Druhý krok se nazývá **inflace**. Matematický zápis je uveden v definici 3.5 [16].

Definice 3.5 *Nechť vstupem je matice $M \in \mathbb{R}^{nxn}$, $M \geq 0$ a nezáporné číslo r. Výsledná matice obsahuje přepočítané hodnoty hran v každém sloupci matice M pomocí inflačního parametru r dle výpočtu níže. Takovouto matici označíme $\Gamma_r M_{pq}$, kde Γ_r se nazývá inflační operátor s inflačním koeficientem r. Formálně zapsáno:*

$$(\Gamma_r M)_{pq} = \frac{(M_{pq})^r}{\sum_{t=1}^{n}(M_{tq})^r}$$

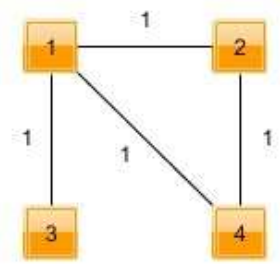
Inflační operátor je zodpovědný jak za zvyšování hodnoty hrany, tak za snižování hodnoty hrany. Silné hrany dále zesílí a slabé hrany ještě zeslabí. Inflační parametr r ovlivňuje míru zesílení/zeslabení a ve výsledku určuje granularitu shluků.

Výsledné hodnoty konvergují opět do sloupcově stochastické matice. Z této matice pak poznáme, které vrcholy patří k sobě (tj. do stejné komunity). Princip rozdělení grafu do komunit je popsán v příkladu dále.

Algoritmus MCL [16]:
Vstup: Neorientovaný graf G, rozšiřovací parametr k a inflační parametr r.
Výstup: Nově ohodnocený graf G prostřednictvím matice M.

1. Vstupní graf reprezentuj pomocí matice sousednosti.
2. Přidej vazbu sám na sebe, tj. přičti jednotkovou matici. Tento bod je volitelný.
3. Normalizuj matici, tj. vytvoř sloupcově stochastickou matici.
4. Rozšiř matici pomocí rozšiřovacího parametru k.
5. Vypočítej inflaci pro všechny hrany.
6. Opakuj body 4 a 5 do splnění podmínky. (Podmínka = další iterace by již nic nezměnila)
7. Z výsledné matice M vyber komunity a vrať je.

Příklad 3.2
Algoritmus je nejlépe pochopitelný na jednoduchém příkladu. Proto si nyní projdeme jednotlivé kroky a ukážeme si jak algoritmus MCL pracuje.



Obrázek 9: Graf pro ukázku výpočtu MCL

Vstupem je graf z obrázku 9, inflační parametr $k = 2$, rozšiřovací parametr $r = 2$

1. Vytvoř matici sousednosti.

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

2. Přidej vazbu sám na sebe, tj. přičti jednotkovou matici. Tento bod je volitelný.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

3. Normalizuj matici, tj. vytvoř sloupcově stochastickou matici.

$$P = \begin{pmatrix} 0.25 & 0.33 & 0.5 & 0.33 \\ 0.25 & 0.33 & 0 & 0.33 \\ 0.25 & 0 & 0.5 & 0 \\ 0.25 & 0.33 & 0 & 0.33 \end{pmatrix}$$

4. Rozšiř matici pomocí rozšiřovacího parametru $k = 2$.

$$P^2 = \begin{pmatrix} 0.35 & 0.31 & 0.38 & 0.31 \\ 0.23 & 0.31 & 0.13 & 0.31 \\ 0.19 & 0.08 & 0.38 & 0.08 \\ 0.23 & 0.31 & 0.13 & 0.31 \end{pmatrix}$$

5. Vypočítej inflaci pro všechny hrany.

$$\Gamma_2 M_{pq} = \begin{pmatrix} 0.47 & 0.33 & 0.45 & 0.33 \\ 0.20 & 0.33 & 0.05 & 0.33 \\ 0.13 & 0.02 & 0.45 & 0.02 \\ 0.20 & 0.33 & 0.05 & 0.33 \end{pmatrix}$$

6. Opakuj body 4 a 5 do splnění podmínky. Podmínkou je, že již nedochází k žádným změnám.

   Postupným iterováním se dostaneme k této matici:

$$M = \begin{pmatrix} 1 & 0.33 & 0.50 & 0.33 \\ - - - & 0.33 & - - - & 0.33 \\ - - - & - - - & 0.50 & - - - \\ - - - & 0.33 & - - - & 0.33 \end{pmatrix}$$

■

Jak vidíme, výpočet konverguje opět ke sloupcově stochastické matici. V tomto případě by k rozdělení na komunity nedošlo, avšak pro ukázku výpočtu je tento malý příklad vhodný. V následujícím příkladě si ukážeme, jak rozpoznat komunity.

Příklad 3.3
Nyní si ukážeme příklad z [6], ve kterém dojde k rozdělení původního grafu do komunit. Na tomto příkladu si vysvětlíme, jak z výsledné matice určíme komunity. Stejně jako na vstupu, i na výstupu opět dostaneme sloupcově stochastickou matici. Některé hrany však konvergují k nule a proto bude mít výsledná matice pouze několik prvků.

$$P = \begin{array}{cccccccccccc}
0.200 & 0.250 & -- & -- & -- & 0.333 & 0.250 & -- & -- & 0.250 & -- & -- \\
0.200 & 0.250 & 0.250 & -- & 0.200 & -- & -- & -- & -- & -- & -- & -- \\
-- & 0.250 & 0.250 & 0.200 & 0.200 & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & 0.250 & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & -- \\
-- & 0.250 & 0.250 & -- & 0.200 & -- & 0.250 & 0.200 & -- & -- & -- & -- \\
0.200 & -- & -- & -- & -- & 0.333 & -- & -- & -- & 0.250 & -- & -- \\
0.200 & -- & -- & -- & 0.200 & -- & 0.250 & -- & -- & 0.250 & -- & -- \\
-- & -- & -- & 0.200 & 0.200 & -- & -- & 0.200 & 0.200 & -- & 0.200 & -- \\
-- & -- & -- & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & 0.333 \\
0.200 & -- & -- & -- & -- & 0.333 & 0.250 & -- & -- & 0.250 & -- & -- \\
-- & -- & -- & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & 0.333 \\
-- & -- & -- & -- & -- & -- & -- & -- & 0.200 & -- & 0.200 & 0.333
\end{array}$$

Stejnými výpočty jako v předchozím příkladu dostaneme z této matice P výslednou (sloupcově stochastickou) matici M:

$$M = \begin{array}{cccccccccccc}
1.000 & -- & -- & -- & -- & 1.000 & 1.000 & -- & -- & 1.000 & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & 1.000 & 1.000 & -- & 1.000 & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & 0.500 & -- & -- & -- & 0.500 & 0.500 & -- & 0.500 & 0.500 \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & 0.500 & -- & -- & -- & 0.500 & 0.500 & -- & 0.500 & 0.500 \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & --
\end{array}$$

Vrcholy si nyní rozdělíme na dva typy. Vrcholy které ostatní přitahují (v angličtině attractors) a vrcholy, které jsou jimi přitahovány. Vrcholy, které ostatní přitahují, poznáme tak, že mají alespoň jednu kladnou hodnotu ve svém řádku výsledné matice. Přitáhnou pak k sobě do komunity ty vrcholy, které mají kladnou hodnotu ve stejném řádku. V příkladu výše nám tedy vzniknou komunity: {1, 6, 7, 10}, {2, 3, 5} a {4, 8, 9, 11, 12}. Čísla vrcholů jsou indexy matice M. V některých případech se výsledné komunity mohou částečně překrývat (tzn. že dvě komunity budou mít jeden společný vrchol). ■

3.2 Lokální shlukovací algoritmy

Na rozdíl od globálních shlukovacích algoritmů, popsaných výše se lokální algoritmy soustředí pouze na určitou část grafu. Nepracují tedy s celým grafem, ale středem zájmu je okolí vybraného vrcholu (či vrcholů). Existuje kolem zvoleného vrcholu shluk? Na tuto otázku nám odpoví právě lokální shlukovací algoritmy, jejichž zástupce si nyní uvedeme.

3.2.1 Lokální detekce komunit pomocí metody Seed Expansion

Algoritmus Seed Expansion [12] se používá pro odhalení shluku kolem určitého vrcholu. Jedná se tedy o zástupce lokálního shlukování (na rozdíl např. od separace pomocí metody NS, která je popsána výše). Algoritmus má na vstupu námi zvolený vrchol z grafu a hledá komunitu kolem tohoto vrcholu. Opět zde pracujeme s ohodnoceným grafem $G = (V, E)$. Nalezená komunita nám vytvoří podgraf C grafu G. Množina všech ostatních vrcholů U leží mimo komunitu C, tedy pro ně platí: $U = \{v | v \in V_G, v \notin V_C\}$, kde V_G je množina všech vrcholů grafu G a V_C je množina všech vrcholů komunity, tj. podgrafu C. Pro výpočty podobností vrcholů si ještě musíme zadefinovat množinu k-tých sousedů vrcholu v_1 jako $N(v_1, k), k \geq 0$. Pokud $k = 0$, pak $N(v_1, 0) = v1$. Pokud $k = 1$, pak $N(v_1, 1) = \{v | (v, v_1) \in$

$E_G, v \in V_G\}$. Pokud $k = 2$, pak $N(v_1, 2) = \{v|(v, v_2) \in E_G, v \in V_G, v_2 \in N(v_1, 1)\}$. Jedná se tedy o kolekci všech sousedů $N(v_1, 1)$, neboli vrcholy, ke kterým se dostaneme pomocí cesty délky maximálně 2.

Jedna z možností, jak vypočítat podobnost vrcholů je výpočet pomocí poměru počtu společných sousedů ku všem sousedům daných vrcholů. Vzorec tedy vypadá takto [12]:

$$Sim(v_1, v_2) = \frac{N(v_1,k) \cap N(v_2,k)}{N(v_1,k) \cup N(v_2,k)}, \text{ kde } k \geq 1$$

Podobnost mezi nově přidávaným vrcholem a komunitou je poté spočtena jako [12]:

$$Sim(v, C) = \frac{N(v,k) \cap N(s,k)}{N(v,k) \cup N(s,k)},$$

kde s je centrálním uzlem komunity C (Seed) a v je nově přidávaný vrchol.

V parametru θ našeho algoritmu nastavíme hranici propustnosti vrcholů do komunity. Pokud bude vypočítaná podobnost ze vzorce větší než námi nastavená hranice (tzv. threshold), pak vrchol přidáme do komunity. Takovýchto vrcholů může být v jednom cyklu samozřejmě více. Tím pádem máme i více kandidátů na centrální vrchol pro příští iteraci. Musíme si tedy nadefinovat, jak budeme vybírat nový centrální vrchol. Tento výpočet převezmeme z [12] a počítá se jako poměr stupně daného vrcholu v rámci komunity C a celkového počtu vrcholů v komunitě C. Vzorec je tedy následující:

$$I(v) = \frac{deg(v,C)}{N-1}$$

Jako nový centrální vrchol bude vybrán ten, jehož hodnota $I(v)$ bude nejvyšší. První centrální vrchol určíme na vstupu algoritmu.

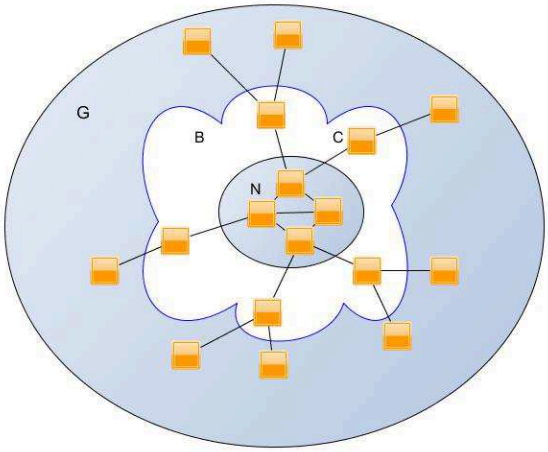
Algroritmus postupně přidává vrcholy do množiny C podle určitého pravidla. Pomocí parametrů můžeme ovlivnit, které vrcholy algoritmus do komunity zařadí a které ne. Běží tak dlouho, dokud může přidávat další vrcholy.

V prvním kroku se tedy vybere vrchol a označí se jako Seed. Tento vrchol je nyní pro komunitu centrální. V dalším kroku se vyberou jeho přímí sousedi jako kandidáti na členy komunity. Z těchto vrcholů se vyberou ty, které jsou centrálnímu vrcholu nejbližší. Poté z těchto nových vrcholů vybereme ten nejdůležitější jako centrální vrchol pro příští iteraci. Centrální vrcholy se neopakují.

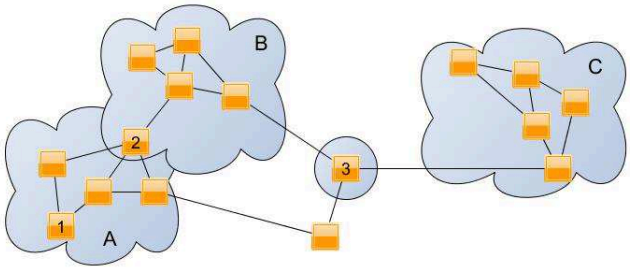
Na obrázku č. 10 je znázorněn výskyt komunity $C = (V', E')$ v daném grafu $G = (V, E)$. Tuto komunitu C tvoří dvě množiny vrcholů B a N, pro které platí:

Oblast B je množinou tzv. hraničních vrcholů komunity. Jedná se o vrcholy, které mají sousedy jak v množině N (množina interních vrcholů komunity), tak mimo komunitu v rámci grafu G. Množina vrcholů N se tedy nazývá interní a má sousedy pouze v komunitě C. Komunita C je součtem množin B a N: $C = B \cup N$ [13].

Není pravidlem, že každý vrchol patří do nějaké komunity. V principu jsou tři možnosti, v jakém stavu se může vrchol vyskytovat v rámci komunit. Tyto možnosti jsou vyobrazeny na obrázku č. 11.



Obrázek 10: Struktura komunity v grafu



Obrázek 11: Možnosti výskytu vrcholu v rámci komunit

První možností je umístění vrcholu do jednoho shluku. Na obrázku se jedná o vrchol č. 1 a shluk A. Druhou možností je, že vrchol patří stejnou vahou do dvou nebo více komunit. Na obrázku se jedná o vrchol č. 2, který patří jak do komunity A, tak do komunity B. Těmto komunitám se říká překrývající (z anglického overlapping, nebo taky cover). Poslední třetí variantou je, že vrchol nepatří do žádné komunity. Na obrázku jde o vrchol č. 3. V tomto případě nám algoritmus nevrátí žádnou komunitu, ale pouze vstupní vrchol (záleží na naprogramování aplikace).

Algoritmus Seed expansion [12]:

Vstup: Neorientovaný ohodnocený graf G, vstupní vrchol s, počet opakování, parametr k, parametr θ

Výstup: Nalezená komunita C

1. Přidej vstupní (centrální) vrchol do komunity C.

2. Do splnění podmínky opakuj (např. zvolený počet opakování):
   (a) Vyber sousedy aktuálního centrálního vrcholu.
   (b) Pro všechny sousedy spočítej jejich podobnost s centrálním vrcholem.
   (c) Jestliže je podobnost větší než θ, přidej vrchol do komunity.
   (d) Pro všechny vrcholy v komunitě spočítej důležitost vrcholů.
   (e) Vrchol s největší hodnotou důležitosti označ jako centrální vrchol pro příští iteraci.
3. Vrať komunitu C.

3.2.2 Lokální detekce komunit pomocí random walku

Většina metod pro lokální detekci komunit není založena na metodě random walk. V případě lokální detekce totiž není problém se rozhlédnout v okolí zvoleného bodu a do určité vzdálenosti jej prozkoumat celé. My zde popíšeme jednu z metod, která random walk využívá a v experimentech provedeme porovnání s metodou Seed expansion, popsanou v předchozí kapitole. Metoda s random walkem je popsána v [17].

Postup detekce lokální komunity dle této metody lze rozdělit do tří kroků [17]:

1. Proveď sérii random walků pro vygenerování startovní množiny komunit C.
2. Množina komunit, inicializovaná v prvním kroku, a jejich vrcholy jsou ohodnoceny pomocí funkce, která je popsána dále.
3. Dle vypočteného hodnocení jsou vrcholy z původních komunit vyměňovány za jiné (z jiných komunit) tak, aby došlo ke zlepšení struktury lokální komunity. Tyto "výměny"vrcholů se provádí buďto dokud velikost nejlepší komunity nedosáhne uživatelem zvolené hodnoty k, nebo již není možné strukturu nejlepší komunity dále vylepšit. Metoda následně vrátí komunitu s nejvyšším ohodnocením.

Budeme zde popisovat postup pro neohodnocený graf G, tzn. že ohodnocení každé hrany se rovná jedné. Pro jiné ohodnocení není problém algoritmus modifikovat. Algoritmus začíná sérií random walků ze zvoleného vrcholu s a prozkoumává neznámé okolí v grafu G, čímž se vygeneruje počáteční množina komunit. Tyto komunity jsou kandidáty na optimální shluk (komunitu) kolem vrcholu s. Tyto komunity jsou tedy tvořeny podmnožinou vrcholů a hran grafu G. Každému vrcholu je přiřazena hodnota γ, která je definována jako [17]:

$$\gamma(u) = \frac{deg(u)\ in\ G}{deg(u)\ in\ new\ community}$$

Jde tedy o podíl stupně vrcholu v původním grafu G a počtu sousedů v nově vytvořené komunitě (neboli stupně vrcholu v nově vygenerované komunitě). Tyto vrcholy jsou poté v rámci své komunity seřazeny dle hodnoty γ od nejvyšší. Vrchol s nejvyšší hodnotou γ má největší pravděpodobnost objevit bezprostřední sousedy v jiných komunitách, proto

je nazýváme *aktivními vrcholy*. Také si musíme uvědomit, že se každý vrchol může objevit ve více komunitách, avšak s jinými hodnotami γ [17].

Podobně jako vrcholy ohodnotíme také jednotlivé komunity, vygenerované random walkem. Každé komunitě tedy přiřadíme hodnotu Γ, která se vypočítá jako průměr hodnot γ dané komunity, jejíž výpočet je uveden v předchozím odstavci. Komunity následně seřadíme dle hodnoty Γ. Z těchto výpočtů také vyplývá, že komunita s vyšší hodnotou Γ má více aktivních vrcholů. Výsledná komunita C by tedy měla odstranit aktivní vrcholy, nebo přidat jeho bezprostřední sousedy, aby měla co nejlepší shlukovou strukturu [17].

Pro dosažení tohoto cíle použijeme iterativní aglomeraci na seřazených komunitách a budeme se tedy snažit snížit hodnoty Γ. Aglomerace probíhá výměnou aktivních vrcholů v rámci vysoce ohodnocených komunit. Je tedy jasné, že komunita obsahující aktivní vrcholy má velkou pravděpodobnost objevit bezprostřední sousedy v jiných komunitách. Tyto výměny se projeví ve změně sousedů daných vrcholů a tedy i ve změně jejich γ hodnot. Tím se samozřejmě změní také ohodnocení komunity Γ. Výsledkem těchto výměn tedy je sbližování jednotlivých vrcholů k sobě a snižování hodnot γ. To vede k nalezení nejlepší možné komunity v okolí vstupního vrcholu. Tento proces může probíhat do získání určitého počtu vrcholů ve výsledné komunitě (tento počet je na vstupu algoritmu), nebo algoritmus zjistil celou lokální strukturu (tedy již nelze pokračovat).

Algoritmus lokální detekce komunit pomocí random walku [17]:

Vstup: Neorientovaný ohodnocený graf G, počáteční centrální vrchol s, počet generovaných komunit η, počet vrcholů v jedné komunitě l

Výstup: Nalezená komunita C

1. Vygeneruj η komunit, každá o délce cesty l.
2. Pro každý vrchol každé komunity vypočítej hodnotu γ.
3. Seřaď vrcholy v rámci každé komunity podle hodnoty γ od nejvyššího.
4. Pro každou komunitu vypočítej hodnotu Γ.
5. Seřaď komunity podle hodnoty Γ od nejvyššího.
6. Do splnění podmínky pro výměnu vrcholů proveď:
   (a) Pro každé dvě po sobě jdoucí komunity proveď výměnu vrcholů, pokud tím dojde ke zlepšení hodnot γ a tudíž také Γ.
   (b) Odeber prázdné komunity.
   (c) Přepočítej hodnoty γ a Γ a znovu seřaď vrcholy a komunity podle nových hodnot.
7. Vyber nejvýše postavenou komunitu.
8. Vrať komunitu C.

4 Implementace

V této kapitole se budeme zabývat vlastní implementací reprezentace grafů a shlukovacích algoritmů. Shlukovací metody, jejichž implementace je zde popsána, jsou uvedeny v předchozí kapitole.

Program je napsán v moderním, objektově orientovaném jazyce C# s aktuálním frameworkem .NET 4.5. Vývoj probíhal v prostředí Visual Studio 2012 od společnosti Microsoft. V prvních fázích implementace byla jako prezentační vrstva použita konzolová aplikace. Pro testování funkčnosti algoritmů prostředí příkazového řádku plně postačuje. Z důvodu uživatelské přívětivosti bylo později nasazeno prostředí Windows Forms (zkráceně WinForms). Jedná se o prostředí klasické desktopové aplikace. Desktopová aplikace byla vybrána z důvodu zaměření aplikace. Webové rozhraní, ani mobilní aplikace nejsou nyní nutné.

4.1 Reprezentace grafu v jazyce C#

Jedna z nejdůležitějších otázek je v tomto programu vhodná datová reprezentace grafů. Výběr a implementace vhodné struktury není triviální záležitostí, jelikož musíme uvažovat s více faktory, které nás budou ovlivňovat. Především je to rozsáhlost dat. Jak již bylo několikrát zmíněno, vstupními daty jsou grafy. Většinou jde o komunitní grafy, tedy grafy reprezentující lidi a různé vazby mezi nimi. Tyto vazby pak mohou mít různou váhu. Obecně máme několik možností, jak grafy v počítači reprezentovat. Nyní si ukážeme některé z nich, které byly postupně použity v našem řešení a řekneme si, proč některé z nich nejsou pro náš případ vhodné.

4.1.1 Dvourozměrné pole

První možností, jak ukládat a pracovat s grafy v počítači, je dvourozměrné pole. Toto řešení napadne téměř každého jako první varianta. V našem případě by se tedy jednalo vždy o čtvercovou matici o velikosti dle počtu vrcholů v grafu. Pokud bychom měli graf o deseti vrcholech (n = 10), naše matice by měla deset sloupců a deset řádků. Jednotlivé prvky matice by pak byly váhy daných hran. Toto řešení je vidět v kapitole č. 2 na obrázku č. 4. Implementace pomocí dvourozměrného pole je však vhodná pouze pro malá data. Pokud máme počet vrcholů $n = 10$, pak musíme uložit n^2 hodnot, tedy 100. Námi zpracovávaná data však mají desítky až stovky tisíc uzlů. Tím se dostáváme k problému uložení 50000^2 hodnot, nebo i více. To je paměťově velice náročné a značně neefektivní, jelikož většina hodnot budou nuly. Při prvním spuštění a pouhém načtení hodnot (alokování paměti) byly zaplněny celé 4 GB paměti, což bylo stále málo. Tento způsob ukládání grafů byl tedy zavrhnut a hledalo se jiné řešení. Nové řešení muselo být vhodnější pro reprezentaci řídkých dat (většina se rovná nule).

4.1.2 Objektově orientovaný přístup

Další přístup, který jsme zkusili, byl objektový. Vytvořily se tedy třídy $Graph$, $Vertex$, $Edge$ a několik dalších pomocných. Tato reprezentace vyřešila paměťovou náročnost, jelikož se ukládaly pouze potřebná data. Ve třídě $Vertex$ se ukládaly informace o jednotlivých uzlích a ve třídě $Edge$ pak informace o hranách. Výhodou je elegantní a přehledný styl implementace. Nevýhoda této reprezentace dat se ukázala později při práci s daty. V tomto tvaru je velice složité počítat maticové operace jako sčítání, odčítání, nebo násobení. Toto řešení bylo tedy opuštěno z důvodu složitosti algoritmů a tím pádem i časové náročnosti.

4.1.3 Knihovna MathNet Numerics

Velice nadějnou možností reprezentace grafů se zdálo být použití knihovny třetí strany s názvem MathNet Numerics. Tato knihovna mimo jiné poskytuje třídu $SparseMatrix$, z jejíhož názvu vyplývá řídká reprezentace dat. Vytvoření a naplnění objektu daty je velice jednoduché. Příklad je vidět ve výpisu kódu 1.

```
public SparseMatrix CreateSparseFromDictionary(Dictionary<int, List<Tuple<int, float>>> dict)
{
    SparseMatrix newMatrix = new SparseMatrix(size);

    foreach (KeyValuePair<int, List<Tuple<int, float>>> pairs in dict)
    {
        foreach (Tuple<int, float> item in pairs.Value)
        {
            newMatrix[pairs.Key - 1, item.Item1 - 1] = item.Item2;
        }
    }
    return newMatrix;
}
```

Výpis 1: Ukázka inicializace třídy SparseMatrix

Třída má také připravené metody pro práci s maticemi. Jako příklad můžeme uvést metody *Add(Matrix<T> other)* pro sčítání, *Clone()* pro vytvoření kopie objektu, *L1Norm()* pro spočítání L1 normy matice, nebo také *Multiply(Matrix<T> other)* pro násobení matic. Všechny tyto připravené metody fungují výborně a mohou nám ušetřit práci s programováním vlastních řešení. Problém je pouze s metodou pro násobení matic. Po naimplementování a spuštění metody využívající právě toto násobení jsme se ani po dlouhé době vykonávání programu nedočkali výsledku. Následně jsme zjistili, že výsledkem této metody již není řídká matice, ale klasická *hustá* matice. Tudíž i toto jinak velice povedené řešení jsme museli opustit a vymyslet vlastní řešení násobení velkých matic, které se nyní stalo hlavní otázkou implementace.

4.1.4 Implementace pomocí slovníku

Hledali jsme tedy řešení, které by vyřešilo všechny problémy předcházejících variant reprezentace grafů v počítači. To se nám podařilo díky několika datovým strukturám, které jsou obsaženy v .NETu 4.5 a tedy v jazyce C#. Potřebovali jsme vlastně seznam vrcholů a pro každý vrchol si udržovat seznam sousedů a ohodnocení dané hrany. Využili jsme tedy datovou strukturu **Dictionary** (česky slovník), která nám umožňuje rychle přistupovat k datům pomocí zvoleného klíče libovolného datového typu. Rychlost přístupu je zajištěna tzv. hashováním a pro výběr jednoho prvku tedy nemusíme procházet celou množinu prvků. Slovník je v .NET C# zastoupen generickou třídou Dictionary s připravenými metodami jako *Add(key, value)* pro přidávání položek do slovníku, *ContainsKey(key)* pro zjištění zda slovník obsahuje položku s daným klíčem, nebo třeba vlastnost *Count*, která vrátí počet položek ve slovníku.

Jelikož jako hodnotu ve slovníku můžeme použít pouze jednu položku (určitého datového typu), museli jsme jako hodnotu použít další datovou strukturu, a to **Tuple**. Jedná se o strukturu n-tic prvků o počtu jednoho až sedmi. Pomocí vnoření však můžeme počet prvků teoreticky rozšířit neomezeně. Nám však budou stačit dvojice. První číslo v této dvojici je identifikátor souseda (datový typ integer) a druhé číslo je ohodnocení dané hrany (datový typ float). Takovýchto k sobě přiřazených dvojic může mít každý vrchol více, proto musíme vytvořit seznam, neboli List těchto dvojic.

Třetí použitou datovou strukturou je tedy **List**. Je použit pro seznam dvojic, přiřazených k jednotlivým vrcholům. V .NET C# je List zastoupen generickou třídou List, která má připravené metody pro práci se seznamy. Pro příklad uveďme třeba *Add()* pro přidávání prvků, *Find* pro hledání prvku podle zadaných podmínek, nebo *RemoveAt* pro odstranění prvku na dané pozici.

Vytvoření takovéto struktury je vidět ve výpisu 2 níže. Kód načítá data ze souboru a naplní jimi výše popsanou strukturu pro pozdější zpracování.

```
// Load File
// @return Graph in dictionary structure
public Dictionary<int, List<Tuple<int, float>>> LoadFile(string filename)
{
    string line = "";  int num1, num2;  float num3;
    Dictionary<int, List<Tuple<int, float>>> dict = new Dictionary<int, List<Tuple<int, float
        >>>();

    using (StreamReader sr = new StreamReader(filename))
    {
        while (sr.Peek() >= 0)
        {
            // Read next line
            line = sr.ReadLine();
            // Parse line into 3 strings
            string [] integerStrings = line.Split(new char[] { ' ', '\t', '\r', '\n' },
                StringSplitOptions.RemoveEmptyEntries);
            Int32.TryParse(integerStrings[0], out num1);
            Int32.TryParse(integerStrings[1], out num2);
```

```
            float.TryParse(integerStrings[2], out num3);
            if (num1 >= size) size = num1;
            // Fill dictionary
            Tuple<int, float> tup = new Tuple<int, float>(num2, num3);
            if (dict.ContainsKey(num1))
            {
                dict[num1].Add(tup);
            }
            else
            {
                List<Tuple<int, float>> l = new List<Tuple<int, float>>();
                l.Add(tup);
                dict.Add(num1, l);
            }
        }
    }
    return dict;
}
```

Výpis 2: Načtení dat ze souboru do připravené struktury

4.2 Implementace metody Neighbor Similarity

Metoda Neighbor Similarity (dále jen NS) byla zpracovávána jako první. Z toho vyplývají jisté potíže na které jsme v průběhu implementace narazili. Některé z nich jsou již zmíněny o kapitolu výše. Jednalo se především o vhodnou reprezentaci řídkých dat pro pozdější zpracování. Není problém naimplementovat metodu NS pro malá data. Problém nastává při výpočtech nad velkými daty. V této metodě se totiž provádí operace násobení nad maticemi. Podle zvolených parametrů to může být i opakované násobení. Stěžejní funkce této metody je tedy násobení, proto si myslím, že je vhodné ji zde uvést (výpis 3):

```
// Multiply matrix
// @return Product of matrices
public Dictionary<int, List<Tuple<int, float>>> Multiply(Dictionary<int, List<Tuple<int, float
    >>> prob, Dictionary<int, List<Tuple<int, float>>> probT)
{
    Dictionary<int, List<Tuple<int, float>>> multiplied = new Dictionary<int, List<Tuple<int,
        float>>>();
    float tempSum = 0;

    foreach (KeyValuePair<int, List<Tuple<int, float>>> pairs in prob)
    {
        foreach (KeyValuePair<int, List<Tuple<int, float>>> pairsT in probT)
        {
            tempSum = 0;
            foreach (Tuple<int, float> item in pairs.Value)
            {
                foreach (Tuple<int, float> itemT in pairsT.Value)
                {
                    if (item.Item1 == itemT.Item1)
```

```
                tempSum += item.Item2 * itemT.Item2;
            }
        }
        Tuple<int, float> tup = new Tuple<int, float>(pairsT.Key, tempSum);
        if (tempSum != 0)
        {
            if (multiplied.ContainsKey(pairs.Key))
            {
                multiplied[pairs.Key].Add(tup);
            }
            else
            {
                List<Tuple<int, float>> l = new List<Tuple<int, float>>();
                l.Add(tup);
                multiplied.Add(pairs.Key, l);
            }
        }
    }
  }
  return multiplied;
}
```

Výpis 3: Násobení řídké matice

Vstupem funkce pro násobení je slovník s daty, jehož struktura je popsána výše a dále transponovaná matice pro zefektivnění násobení. Prochází se postupně celý původní slovník a celý transponovaný slovník. V klasickém násobení matic je nutné projít tuto strukturu ještě jednou a z toho vyplývá časová náročnost $O(n^3)$. Díky vhodně zvolené struktuře jsme ale zredukovali časovou náročnost na $O(n^2 * degree)$, kde degree je konstanta, vyjadřující maximální stupeň vrcholů daného grafu. Místo procházení n prvků tedy projdeme jen sousedy daného vrcholu. Úspora času nad velkými daty je tedy značná.

Kromě násobení matic obsahuje příslušná třída také funkce jako CreateProbs(dict) pro vytvoření matice sousednosti, kde prvky jsou pravděpodobnostmi přechodů ke svým sousedům, nebo již zmíněná metoda Transpone(dict) pro vytvoření transponované matice. Dále metody GetSum(dict1, dict2), která provede efektivní sčítání řídkých matic, CreatePkVisitMatrix(prob, probT, k) pro vytvoření Pk vektorů a konečně metoda CreateNewWeightsUsingExp(dict, prob, k) pro nové ohodnocení hran. Pak již data projdou pouze metodou DeleteSmallWeightedEdges(dict, limit) pro odstranění hran s malým ohodnocením.

4.3 Implementace metody Markov cluster algorithm

Implementace metody Markov cluster algorithm je docela podobná implementaci metody Neighbor Similarity. Taktéž zde musíme počítat násobení matic (několikanásobné) a sčítání matic. Popíšu zde tedy pouze metodu pro inflaci (inflace je popsána v kapitole o Markov cluster algorithm). Vstupem této metody je opět slovník s řídkou maticí (normovaná matice sousednosti) a také inflační parametr r. Pro vstupní matici si vypočteme matici transponovanou. Tento algoritmus totiž počítá se sloupcově stochastickou

maticí. Tu získáme právě transponováním matice vstupní, která je řádkově stochastická. Dále už měníme hodnoty hran podle nastaveného inflačního parametru. Část kódu, která zpracovává inflaci, je vidět ve výpisu 4 níže:

```
// MCL
// @return Inflated matrix
foreach (var tuple in dict)
{
    foreach (var item in tuple.Value)
            {
                    sumOfLine = 0;
                    foreach (var itemT in dictT[item.Item1])
                    {
                        sumOfLine += (float)Math.Pow(itemT.Item2, r);
                     }
             Tuple<int, float> tup = new Tuple<int, float>(item.Item1, (float)Math.Pow(item.Item2,
                   r) / sumOfLine);
             if ( inflated .ContainsKey(tuple.Key))
             {
                        inflated [tuple.Key].Add(tup);
             }
             else
             {
                          List<Tuple<int, float>> l = new List<Tuple<int, float>>();
                          l.Add(tup);
                          inflated .Add(tuple.Key, l);
                    }
    }
}
```

Výpis 4: Hlavní část algoritmu MCL

4.4 Implementace metody Lokální detekce komunit pomocí random walku

Implementaci lokální detekce komunit pomocí metody random walk lze rozdělit do dvou částí. První částí je generování náhodných komunit (tj. generování náhodných procházek - random walks) a druhou částí pak výměna vrcholů v těchto komunitách za účelem nalezení nejlepšího shluku. Opět zde uvedeme pouze část této implementace. Konkrétně popíšeme metodu z první jmenované části - generování náhodných komunit. Metoda $GenerateCommunities(dict, seedNode, numCommunities, rw_length)$ má čtyři vstupní proměnné. Jedná se o původní (celý) graf, počáteční vrchol hledané komunity, počet vygenerovaných komunit a maximální délka každé náhodné procházky. Tato metoda má dvě pomocné funkce. $ChooseNext(node, previous)$ je funkce pro výběr dalšího vrcholu náhodné procházky s dvěma parametry. Jedním je vrchol, ve kterém se aktuálně nacházíme a druhým je seznam navštívených vrcholů. Vrcholy se v rámci jedné procházky nemohou opakovat. Pokud není jiná možnost, procházka se ukončí. Druhou pomocnou funkcí je pak funkce $MultiplyByRandomAndDegree(item)$, která slouží pro výpočet pravděpodobností dalšího přechodu k sousedním vrcholům. V této funkci je také

umístěn onen zmíněný náhodný prvek, který se vynásobí se stupněm daného vrcholu a váhou příslušné hrany. Hlavní část generování náhodných procházek je vidět ve výpisu 5.

```
// Local RW
// @return Initialized communities
while(m < numCommunities)
{
    int node = seedNode;
    List<int> previous = new List<int>();
    Dictionary<int, List<Tuple<int, float>>> community = new Dictionary<int, List<Tuple<int,
        float>>>();
    List<Tuple<int, float>> sl = new List<Tuple<int, float>>();
    community.Add(node, sl);
    for (int i = 0; i < rw_length; i++)
    {
        Tuple<int, float> tup = ChooseNext(node, previous);
        previous.Add(node);
        if (tup != null) {
            if (community.ContainsKey(node)){
                if (!community[node].Contains(tup)){
                    community[node].Add(tup);
                }else{   i--; }
            } else {
                List<Tuple<int, float>> l = new List<Tuple<int, float>>();
                l.Add(tup);
                community.Add(node, l);
            }
            node = tup.Item1;
        }
    }
    communities.Add(community);
    m++;
}
return communities;
```

Výpis 5: Hlavní část metody pro generování random walků.

4.5 Implementace metody Seed Expansion

Metoda Seed Expansion má na vstupu celkem čtyři proměnné. Jedná se o vstupní vrchol, kolem kterého budeme hledat komunitu, počet opakování algoritmu a dále parametry k a θ. Těmito parametry ovlivníme, které vrcholy se do komunity dostanou a které ne. Princip je popsán v kapitole Seed Expansion.

Do zvoleného počtu opakování tedy budeme provádět následující operace. Pomocí metody $GetDictNeighbors(dict, actualSeed)$; zjistíme seznam sousedů aktuálního centrálního prvku a poté v cyklu počítáme podobnosti dle uvedených vzorců. Pokud je podobnost větší než nastavená hodnota θ, přidáme vrchol do komunity. Následně vypočítáme nový centrální prvek a postup opakujeme. Výpočet pro zvolení nového centrálního prvku se provádí v metodě $GetImportance(tuple.Key, community)$;. V této metodě se pro ka-

ždý vrchol spočítá míra důležitosti daného vrcholu v rámci komunity. Vzorec je uveden v teoretické části. Jednoduchým porovnáváním poté zjistíme vrchol s nejvyšší hodnotou této "důležitosti". Pokud nemůžeme nalézt nový centrální prvek (z důvodu, že může být každý vrchol centrálním vrcholem pouze jednou) algoritmus se ukončí. Ve výpisu 6 je uvedena pouze část hlavní metody algoritmu:

```
// Seed Expansion
// @return Comunity
while (i < runTimes) {
    neighbors = GetDictNeighbors(dict, actualSeed);
    foreach (var tuple in neighbors) {
        float similarity = GetSimilarity(tuple.Key, actualSeed, dict, k);
        // If similarity is higher than theta, add node to community
        if ( similarity >= theta) {
            if (!community.Any(w => w.Key == tuple.Key))
                community.Add(tuple.Key, tuple.Value);
        }
    }
    seedNodes.Add(actualSeed);
    float maxImportance = -1;
    int seedCandidate = -1;
    // Search for new Seed
    foreach (var tuple in community)
    {
        // Node hasnt been seed yet
        if (!seedNodes.Any(w => w == tuple.Key))
        {
            float actualImportance = GetImportance(tuple.Key, community);
            if (actualImportance > maxImportance) {
                maxImportance = actualImportance;
                seedCandidate = tuple.Key;
            }
        }
    }
    // new Seed found?
    if (maxImportance >= 0) {
        actualSeed = seedCandidate;
    }
    else
    {
        break;
    }
    i++;
}
```

Výpis 6: Hlavní část algoritmu SE

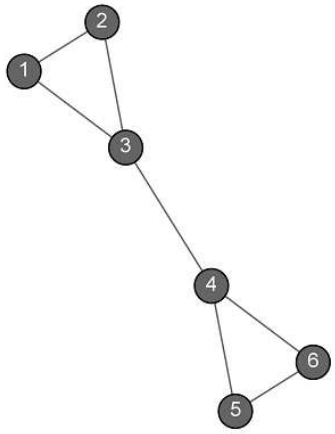
5 Experimenty

5.1 Data pro experimenty

Pro implementaci a testování algoritmů jsem potřeboval nějaká vhodná data menšího rozsahu. Není totiž praktické zkoušet si násobení matic na velkých datech, kde je velikost matice v desítkách či stovkách tisíc. Následně jsem pro experimenty, pro které jsem vybral známé datové kolekce. Menší kolekci Zachary karate club a větší kolekci DBLP. Základní jednotkou času pro všechny experimenty jsou sekundy. U časů nad 59 sekund použijeme formát *minuty:sekundy*.

5.1.1 Malá data pro testování při implementaci

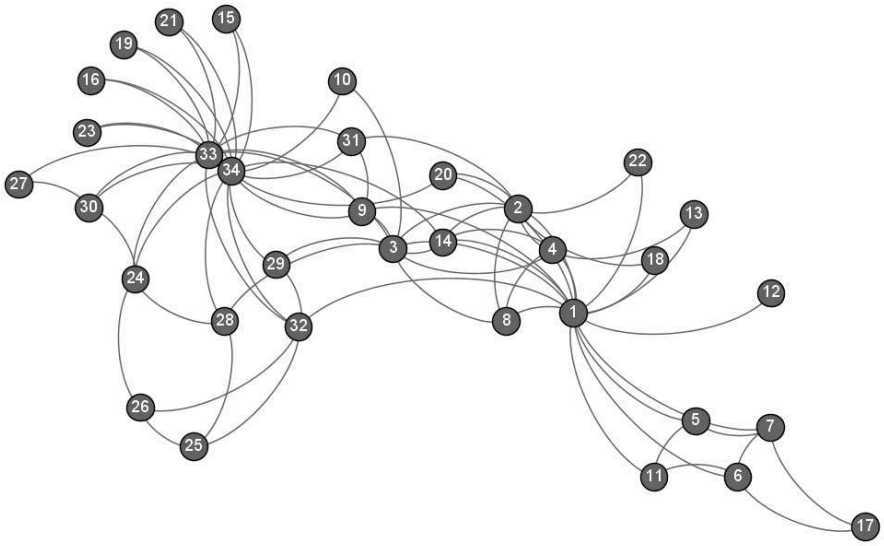
Jak je již popsáno v implementaci, na začátku bylo potřeba vyřešit násobení velkých matic v nějakém rozumném čase. Jelikož bylo vyzkoušeno několik metod násobení matic, bylo potřeba kontrolovat správnost výsledků. Takovou kontrolu lze snadno provést na matici, jejíž produkt (vynásobení matice sama sebou) můžeme rychle spočítat i ručně na papír. Zvolili jsme tedy graf o šesti vrcholech a sedmi hranách. Z takového grafu dostaneme malou matici, kde je většina prvků nulová. Graf této matice je zobrazen na obrázku 12. Jedná se o graf neorientovaný a neohodnocený.



Obrázek 12: Malý graf pro testování při vývoji.

5.1.2 Zachary karate klub

Velmi dobře známá síť Zachary karate klub. Data této kolekce byla získána od členů univerzitního karate klubu v roce 1977. Autorem je Wayne Zachary [19]. Každý vrchol zde reprezentuje člena klubu a každá hrana reprezentuje vazbu mezi těmito členy. Tato kolekce je často používaná v různých literaturách. Jedná se o malou síť, která čítá 34 vrcholů a 78 hran. Jedná se o graf neorientovaný a neohodnocený. Prohlédnout si jej můžeme na obrázku 13.



Obrázek 13: Graf sítě Zachary karate club.

5.1.3 DBLP

Datová kolekce DBLP je stěžejní datovou kolekcí pro tuto práci. Jedná se o bibliografii počítačových věd, nebo také často nazýváno jako databáze spoluautorů. Tato databáze je provozována v Německu od roku 1993. Aktuálně čítá více než 2,3 milionů článků. Každý vrchol reprezentuje autora nějakého článku. Hrany pak reprezentují počet společných článků (knih) mezi dvěma danými autory. Pokud dosud spolu dva autoři nepublikovali, není mezi nimi hrana. Jedná se tedy o neorientovaný graf, ale na rozdíl od předchozích je ohodnocený. Pro naše experimenty použijeme verzi DBLP z roku 1990, která čítá cca 19 tisíc vrcholů a 25 tisíc hran. Důvodem je menší paměťová a časová náročnost a také možnost rozumě zobrazit výsledky. Z důvodu velkého počtu izolovaných vrcholů pro experimenty vybereme pouze největší souvislou komponentu, která obsahuje 1965 vrcholů a 5150 hran.

5.1.4 High Energy Physics - Theory collaboration network

Pro zajímavost byly vybrány i větší datové kolekce. Hlavním důvodem je změření časové náročnosti na větších kolekcích a ověření funkčnosti algoritmů na velkých datech. Tato datová kolekce je obsahem podobná DBLP. Je to také databáze spoluautorů z oblasti fyziky. Data byla sbírána v období mezi lety 1993 a 2003. Celkem obsahuje 9877 vrcholů a 25998 hran. Tato síť je téměř souvislá - největší komponenta obsahuje 8638 vrcholů a 24827 hran. Do testu tedy opět vybereme největší komponentu. Data byla stažena z [20].

Zvolili jsme algoritmus Neigbor similarity s parametry: k = 2, počet iterací = 2 a treshold = 5. Zpracování trvalo 9 minut a 14 sekund. Bylo nalezeno 1631 komunit. Modularita vyšla 0,49 a hustota 0,56.

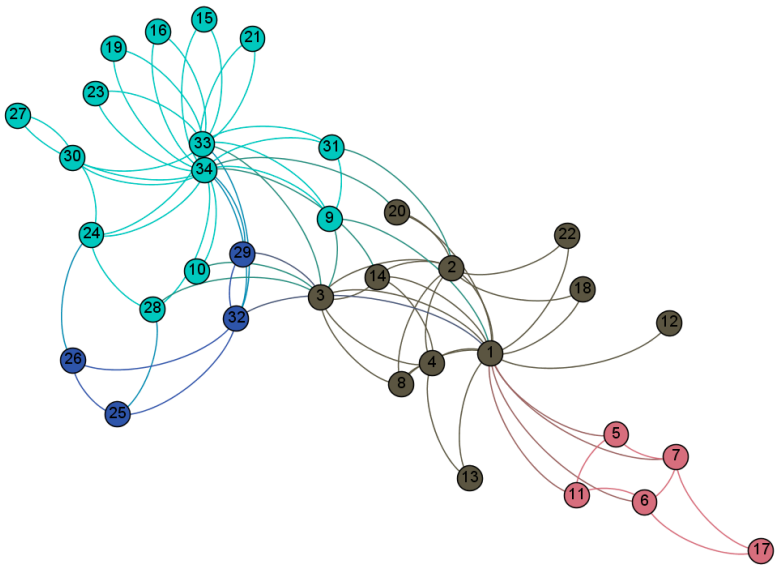
5.1.5 Enron email network

Největší datová kolekce, nad kterou jsme provedli experiment, se jmenuje Enron email a obsahuje emailovou komunikaci mezi uživateli. Obsahuje 36692 vrcholů a 183831 hran. Podobně jako předchozí datová kolekce, je i tato z více než 90% tvořena jednou souvislou komponentou. Ta obsahuje 33696 vrcholů a 180811 hran. Data byla stažena z [21].

Zpracování takto velkých dat se již nepočítá v rámci minut, ani hodin. Je to otázka spíše dnů. Na této kolekci jsme si proto ověřili, že i když používáme řídkou reprezentaci dat, nejsou tyto algoritmy vhodné pro velmi rozsáhlé data. S velikostí dat totiž roste časová náročnost exponenciálně. V následujících experimentech se budeme, z důvodu časové náročnosti, zabývat pouze kolekcemi Zachary karate klub a DBLP.

5.2 Experimenty globálního shlukování

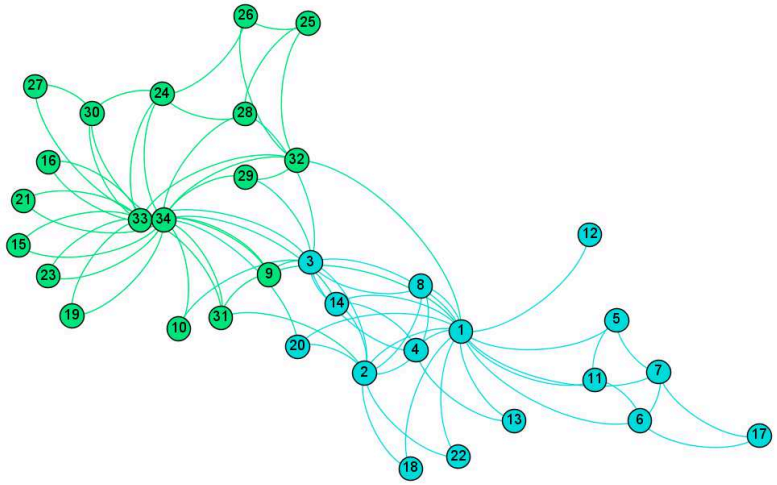
V tabulce 1 jsou vidět výsledky experimentů na datové kolekci Zachary karate club pomocí metody Neighbor similarity. V experimentech měníme hodnoty k, což je parametr algoritmu Neighbor similarity, počet iterací programu a pro několik hodnot také treshold. Pokud hodnotu tresholdu změníme jen nepatrně, na výsledcích se to vůbec nemusí projevit. Pokud velmi změníme treshold, můžeme dostat jiné výsledky. Tyto změny jsou dány tím, že například odebereme více hran (nemusí to však znamenat nalezení více komunit). Jelikož metoda Neigbor similarity konverguje, dostáváme po určitém počtu iterací stejné výsledky. To je vidět v tabulce 1 na posledních dvou řádcích, kdy už se počet komunit, modularita, ani hustota nezměnila. Zároveň se jedná o nejlepší dosažitelnou hodnotu pro tuto datovou kolekci. Modularitu 0,415 dostaneme také jako výsledek v programu Gephi.



Obrázek 14: Ukázka výsledku algoritmu NS pro zachary karate klub s parametry: k: 2, počet iterací: 4, treshold: 4.

k	Počet iterací	Treshold	Počet komunit	Čas zpracování	Modularita	Density
2	2	4	2	0,01	0,043	0,375
2	2	2	1	0,01	-0,064	0,130
2	4	4	4	0,03	0,389	0,487
2	6	4	6	0,04	0,379	0,476
2	8	4	6	0,05	0,379	0,476
3	2	4	1	0,03	-0,064	0,130
3	5	4	3	0,08	0,107	0,477
3	6	4	4	0,1	0,415	0,492
3	9	4	4	0,15	0,415	0,492

Tabulka 1: Tabulka s výsledky algoritmu NS pro Zachary karate klub.



Obrázek 15: Ukázka výsledku algoritmu MCL pro zachary karate klub s parametry: exp. p.: 2, infl. p.: 2, treshold: 0,01.

V tabulce 2 jsou vidět výsledky experimentů pro stejnou datovou kolekci metodou Markov cluster algorithm. U této metody máme taktéž parametry. Jsou to parametry expanční a inflační (které jsou popsány v příslušné kapitole) a dále také treshold. Dále má na algoritmus vliv počet námi nastavených iterací. Tento počet iterací musí být dost velký, abychom získali relevantní výsledky (tj. hodnoty stihnou zkonvergovat), ale zároveň ne moc velký z důvodu časové náročnosti. S každou další iterací totiž provádíme náročnou operaci násobení matic. Pro naše experimenty jsme zvolili pět iterací. Ukázky výsledných komunit z obou algoritmů si můžeme prohlédnout na obrázcích 15 a 14.

V tabulce 3 jsou výsledky pro datovou kolekci DBLP metodou Neighbor Similarity. Treshold byl pro celý test nastaven na hodnotu 0,5. Pouze na druhém řádku tabulky je hodnota tresholdu 2. Můžeme si všimnout obrovského rozdílu v počtu nalezených komunit. Při tresholdu 0,5 se původní graf na komunity vůbec nerozpadl, naopak při tresholdu 2 se rozpadl na 19 komunit. Nalezení správné hranice pro různá data může být

Exp. p. r	Infl. p. k	Treshold	Počet komunit	Čas zpracování	Modularita	Density
2	1	0,01	1	0,120	-	0,139
2	2	0,01	2	0,112	0,371	0,252
2	3	0,01	5	0,006	0,364	0,308
2	3	0,001	4	0,012	0,359	0,385
2	4	0,01	7	0,007	0,340	0,517

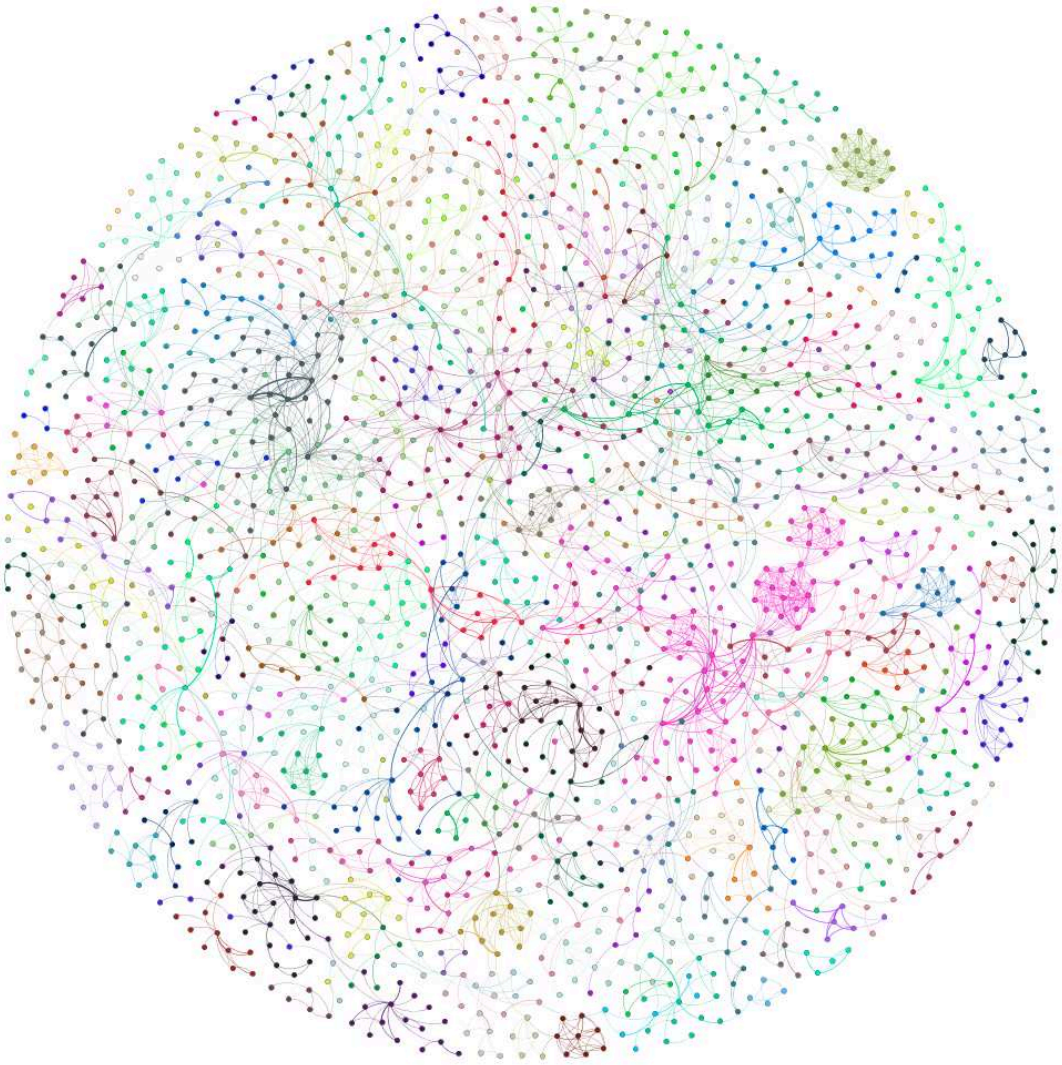
Tabulka 2: Tabulka s výsledky algoritmu MCL pro Zachary karate club.

k	Počet iterací	Treshold	Počet komunit	Čas zpracování	Modularita	Density
2	1	0,5	1	4,835	0	-
2	1	2	19	4,800	0,165	0,459
2	2	0,5	33	9,428	0,287	0,574
2	3	0,5	126	14,685	0,597	0,649
2	4	0,5	233	22,031	0,759	0,698
3	1	0,5	1	18,059	0	-
3	2	0,5	26	37,737	0,300	0,497
3	3	0,5	111	1:01,5	0,684	0,565
3	4	0,5	224	1:25,9	0,765	0,664
3	6	0,5	368	2:04,9	0,746	0,723

Tabulka 3: Tabulka s výsledky algoritmu NS pro největší komponentu z kolekce DBLP 1990.

Exp. p. r	Infl. p. k	Treshold	Počet komunit	Čas zpracování	Modularita	Density
2	2	0,01	354	14,92	0,769	0,668
2	3	0,01	497	9,98	0,670	0,701
2	4	0,01	583	8,80	0,616	0,629
3	2	0,01	218	57,84	0,843	0,504
3	3	0,01	287	33,05	0,805	0,604
3	4	0,01	317	26,73	0,784	0,639
4	2	0,01	169	2:53,47	0,868	0,435
4	3	0,01	244	1:27,89	0,830	0,545
4	4	0,01	274	1:09,33	0,811	0,581

Tabulka 4: Tabulka s výsledky algoritmu MCL pro největší komponentu z kolekce DBLP 1990.

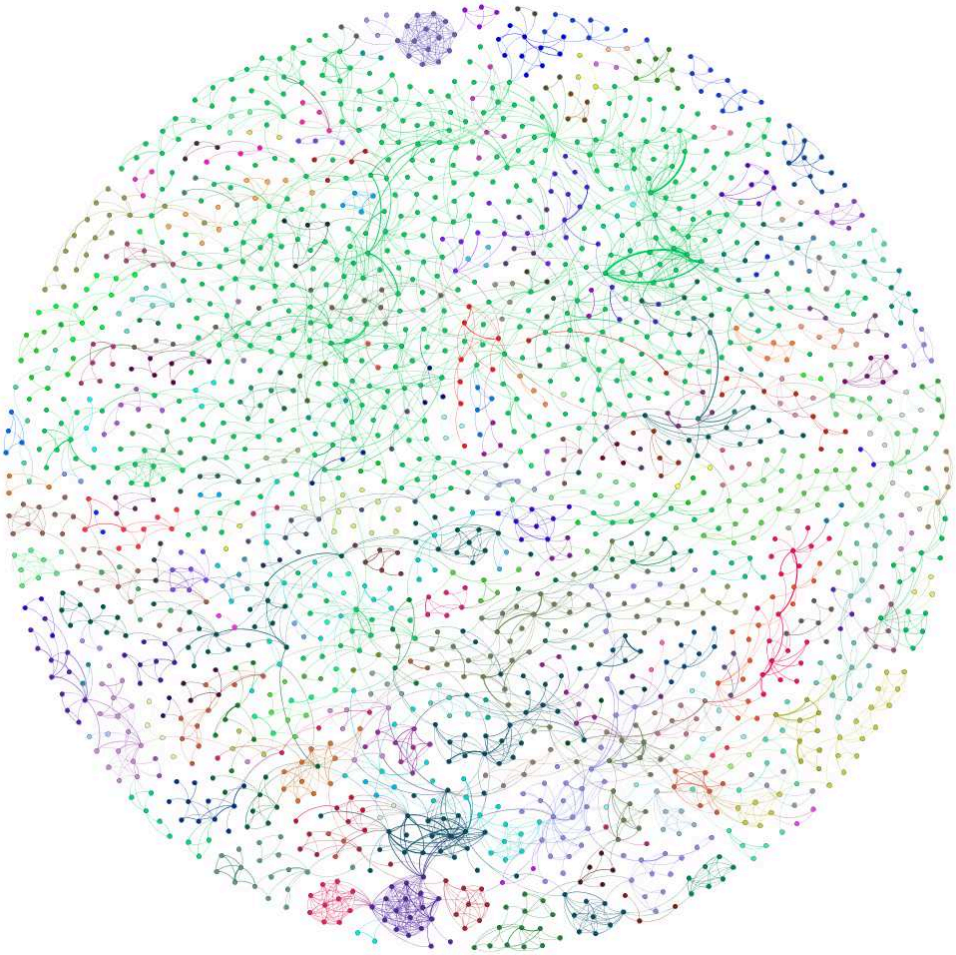


Obrázek 16: Ukázka výsledku algoritmu MCL pro největší komponentu DBLP s parametry: exp. p.: 3, infl. p.: 2, treshold: 0,01.

rozhodující pro správné rozdělení grafu do komunit. Důležité hrany by měly dosahovat vyšších hodnot, nepodstatné hrany by naopak měly být ohodnoceny pod touto hranicí. Taulka 4 obsahuje výsledky pro kolekci DBLP metodou Markov cluster algorithm. Zde byla testováním zvolena jako optimální hodnota tresholdu 0,01. U obou testů si také můžeme všimnout velkých časových rozdílů mezi jednotlivými experimenty. Tyto rozdíly jsou dány počtem násobení matic. Násobení matic, spolu s ohodnocováním hran, je časově nejnáročnější částí těchto algoritmů. Jako optimální hodnota parametru k se ukázala být hodnota 3. Pro hodnotu 2 se graf rozpadá na velké množství komunit, což je dáno

krátkým random walkem. Pro hodnotu 4 jsou sice výsledky nejlepší, ale rapidně narůstá čas zpracování. Hodnoty však nejsou o tolik odlišné.

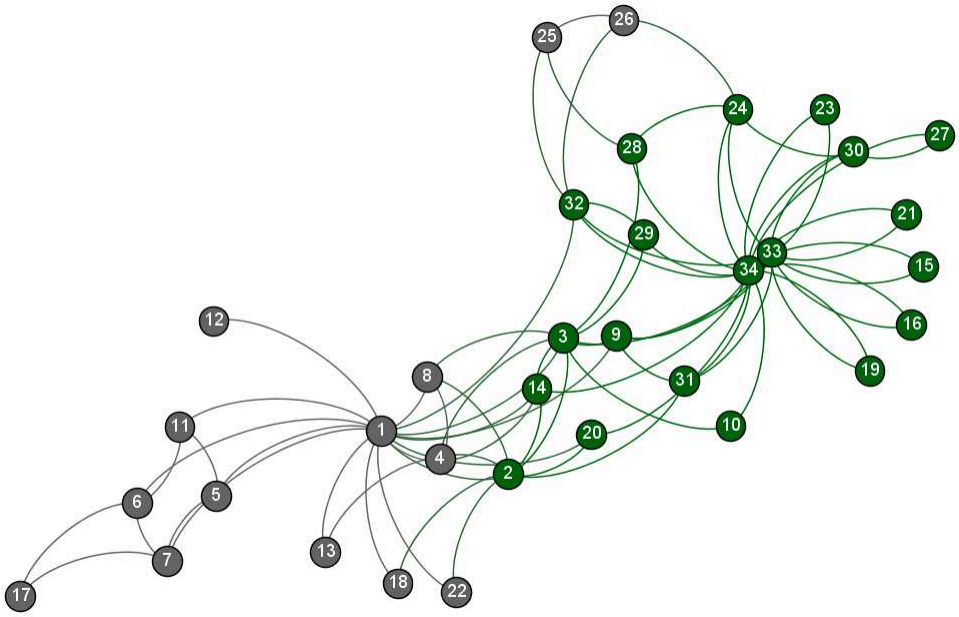
U algoritmu MCL jsme původně odstraňovali malé hrany až na konci samotného algoritmu. Pro rychlejší běh programu jsme algoritmus upravili tak, že malé hrany odstraňujeme při každé iteraci. Dojde tím k daleko menšímu počtu výpočtů při násobení matic. V některých případech je tato časová úspora až desetinásobná. Algoritmem MCL jsme u této datové kolekce dosáhli o něco lepších výsledků. Nejlepší dosažitelný výsledek modularity pro tato data je dle Gephi 0,916. Ukázky výsledných komunit jsou vidět na obrázcích 16 a 17.



Obrázek 17: Ukázka výsledku algoritmu NS pro největší komponentu DBLP s parametry: k: 2, počet iterací: 4, treshold: 0,5. Můžeme si všimnout, že jsme nalezli jiné komunity, než s algoritmem MCL.

5.3 Experimenty lokálního shlukování

V této části práce jsou uvedeny experimenty lokálního shlukování. V tabulce 5 jsou uvedeny výsledky algoritmu Seed expansion pro data Zachary karate klub. Stejně jako globální algoritmy i ty lokální konvergují. To můžeme vidět v tabulce 5 na třetím a čtvrtém řádku. Od určitého počtu iterací se již nalezená komunita nemění. Podobný vliv má i treshold, což vidíme v dolní části tabulky. Postupným snižováním tresholdu dostáváme větší komunitu, avšak po dosažení určité hranice se komunita již nemění. Na obrázku 18 je příklad nalezené komunity kolem vrcholu číslo 34. Volíme tedy počáteční vrchol, počet iterací programu, parametr k a parametr θ.



Obrázek 18: Ukázka komunity v grafu Zachary karate club. Byla použita metoda Seed expansion s parametry: k = 1, počet iterací = 16, treshold = 0,1.

V tabulce 6 jsou uvedeny výsledky algoritmu seed expansion pro data DBLP. Opět volíme parametry jako v předchozím případě. Počáteční vrcholy volíme 1428. Také zde vidíme na třetím a čtvrtém řádku, že algoritmus konverguje. Pro větší hodnoty k dojde k ustálení hodnot později.

V tabulce 7 jsou výsledky algoritmu pro lokální detekci komunit pomocí random walku pro Zachary karate klub. Tabulka 8 pak obsahuje výsledky téhož algoritmu pro největší komponentu z databáze DBLP. Také tento algoritmus konverguje, což je vidět v tabulce 7 na posledních dvou řádcích. Z hodnot je patrné, že tyto dva algoritmy naleznou jiné komunity. V některých případech však mohou vykazovat stejné výsledky, jak je vidět na prvním řádku tabulky 6 a druhém řádku tabulky 8.

Počáteční vrchol	Parametr k	Počet iterací	Parametr θ	Velikost komunity	Density
34	1	2	0,1	6	0,733
34	1	7	0,1	14	0,384
34	1	16	0,1	22	0,221
34	1	25	0,1	22	0,221
34	2	5	0,8	5	0,400
34	2	5	0,7	13	0,282
34	2	5	0,6	22	0,221
34	2	5	0,2	22	0,221

Tabulka 5: Tabulka s výsledky algoritmu seed expansion pro Zachary karate club.

Počáteční vrchol	Parametr k	Počet iterací	Parametr θ	Velikost komunity	Density
1428	1	1	0,2	8	0,678
1428	1	2	0,2	10	0,644
1428	1	5	0,2	12	0,515
1428	1	10	0,2	12	0,515
1428	2	1	0,2	18	0,307
1428	2	2	0,2	18	0,307
1428	2	5	0,2	19	0,280
1428	2	10	0,2	30	0,151

Tabulka 6: Tabulka s výsledky algoritmu seed expansion pro DBLP 1990.

Počáteční vrchol	Počet RW	Délka RW	Velikost komunity	Density
34	4	2	4	0,754
34	4	4	10	0,458
34	4	6	18	0,236
34	4	8	18	0,236

Tabulka 7: Tabulka s výsledky algoritmu pro detekci lokálních komunit pomocí random walku pro Zachary karate club.

Počáteční vrchol	Počet RW	Délka RW	Velikost komunity	Density
1428	5	4	5	0,714
1428	5	6	8	0,678
1428	5	8	14	0,496
1428	5	12	16	0,442

Tabulka 8: Tabulka s výsledky algoritmu pro detekci lokálních komunit pomocí random walku pro DBLP.

6 Závěr

V této práci jsme se seznámili s možnostmi, jak shlukovat objekty v rozsáhlých datových kolekcích. Zpracovávaná data mohou vyjadřovat různé objekty a situace. Popsali jsme si jak metody globální, tak metody lokální. Obě globální metody využívají metodu random walk. Ověřili jsme si, že i algoritmy s náhodným faktorem mohou vykazovat dobré výsledky. Z lokálních algoritmů byla vybrána metoda využívající random walk a pro porovnání také metoda klasická, bez random walku. Z experimentů vyplynulo, že naše algoritmy generují shluky podobných kvalit, jako třeba již zavedený software Gephi. V globálních algoritmech jsme řešili problém časové náročnosti násobení matic, který se nám podařilo snížit na $O(n^2 * deg(n)$ z původních $O(n^3)$, což je pro velké datové kolekce veliká úspora.

Výstupem této práce je přehled několika algoritmů, které můžeme využít pro hledání komunit v rozsáhlých sítích. Některé z těchto algoritmů byly naimplementovány a byly na nich provedeny experimenty. Při tomto experimentování byly některé algoritmy dále upraveny pro menší časovou náročnost. Ani tak však nejsou vhodné pro velmi rozsáhlá data z důvodu náročných operací jako násobení matic či přeohodnocování hran.

Tato práce by se mohla dále rozšířit například v optimalizaci algoritmů (paralelizace, násobení matic pro vícejádrové procesory), nebo v porovnání s dalšími shlukovacími algoritmy, které random walk nevyužívají.

Osobní přínos této práce je značný. Jak v prohloubení znalostí v oblasti teorie grafů, tak v oblasti shlukovacích algoritmů. Velkým přínosem a zkušeností byla také implementace, ve které jsme se potýkali s různými problémy. Úspěšně jsme navrhli vhodnou datovou strukturu a naimplementovali algoritmy tak, aby měly rozumnou paměťovou i časovou náročnost.

7 Reference

[1] Vojtěch Hordejčuk, *Teorie grafů* 2013: http://voho.cz/wiki/matematika/graf/.

[2] David Harel and Yehuda Koren, *On Clustering Using Random Walks.* Dept. of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel 2001.

[3] Bruce Rogers, *Graph cluster with random walks*, presentation, 2010.

[4] Francois Fouss, Alain Pirotte, Jean-Michel Renders, Marco Saerens, *Random-walk computation of similarities between nodes of a graph, with application to collaborative recommendation*, 2006.

[5] Hélio Almeida, Dorgival Guedes, Wagner Meira Jr., Mohammed J. Zaki, *Is there a best quality metric for graph clusters?* Universidade Federal de Minas Gerais, MG, Brazil, Rensselaer Polytechnic Institute, NY, USA, 2011.

[6] Stijn van Dongen, *Graph clustering by flow simulation*, ISBN 90–393–2408–5, Wiskunde en Informatica Proefschriften, Netherlands, 2000.

[7] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, Dorothea Wagner *On Modularity Clustering?* Univ. of Konstanz, Konstanz, 2008.

[8] Santo Fortunato, *Community detection in graphs*, Complex Networks and Systems Lagrange Laboratory, ISI Foundation, Viale S. Severo 65, 10133, Torino, Italy, 2010.

[9] Marcela Henzlová, *Bakalářská práce, Stochastické matice*, Brno, 2007.

[10] Jiří Kučera, *Shluková analýza*, http://is.muni.cz/th/172767/fi˙b/5739129/web/web/main.html, 2013.

[11] Hana Švihálková, *Diplomová práce, Aplikace shlukovacích metod na data klinických registrů*, Brno, 2011.

[12] Bingying Xu, Zheng Liang, Yan Jia, Bin Zhou, Yi Han, *Local Community Detection Using Seeds Expansion.* Cloud and Green Computing (CGC), Second International Conference, 2012.

[13] Mahadevan Vasudevan, Narsingh Deo, *Community Identification algorithm using relative edge density measure.* School of EECS, University of Central Florida, Orlando, Florida, USA, 2010.

[14] Daniel A. Spielman, Department of Computer Science, Program in Applied Mathematics, Yale University, Shang-Hua Teng, Department of Computer Science, Boston University *A Local Clustering Algorithm for Massive Graphs and its Application to Nearly-Linear Time Graph Partitioning*, 2008.

[15] Stijn van Dongen, *Graph clustering by flow simulation.* Wiskunde en Informatica Proefschriften, 2001.

[16] Kathy Macropol, *Clustering on Graphs: The Markov Cluster Algorithm (MCL)* presentation 2012.

[17] G.S. Thakur, R. Tiwari, M.T. Thai, S.-S. Chen - CISE, University of Florida, Gainesville, FL, USA, A.W.M. Dress - 2CAS-MPG Partner Institute for Computational Biology, Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, Shanghai, People's Republic of China, *Detection of local community structures in complex dynamic networks with random walks*, 2007.

[18] Graf komunitní sítě: http://mindthis.ca/wp-content/uploads/2011/09/social-networks-noise.gif, 2011.

[19] Data Zachary karate club: http://konect.uni-koblenz.de/networks/ucidata-zachary, Wayne Zachary, 1977.

[20] Data High Energy Physics - Theory collaboration network: http://snap.stanford.edu/data/ca-HepTh.html, 2003.

[21] Data Enron email network: http://snap.stanford.edu/data/email-Enron.html, William Cohen, 2009.